Project Proposal: Splay Trees

Data Structure Description

The data structure that we will implement in C++ in this project are splay trees. They are a type of binary search tree that is self-adjusting. In this data structure, recently accessed elements are moved closer to the root due to the principles of temporal locality. They are moved in this manner because recently accessed elements are likely to be used again in the near future. Furthermore, splay trees work with a few splay rotations: zig, zag, zig-zig, zag-zag, zig-zag, and zag-zig. All of these rotations work to balance the tree and make it self-adjusting; each time a node is accessed the tree performs one of these operations as needed. Furthermore, most splay tree operations have a runtime of O(log N), making it a relatively fast data structure. Splay trees do not require additional memory for balancing information, and are commonly used for items such as caches and databases due to their access patterns and self-adjustments.

Academic Reference

Memory Access Analysis and Optimization Approaches on Splay Trees

Above is the link to the academic paper outlining the performance and structure of splay trees. Because it is an embedded pdf, we are unable to link direct pages. However, below are some relevant sections and pages that define the data structure well:

- Page 1: Introduction
- Page 2: Figure 1 Bottom-up splaying steps
- Page 2: Figure 2 Top-down splaying step

These pages show the structure of splay trees, how they work, as well as some use cases for rotations/splays for these splay trees. Furthermore, this paper talks about how splay trees are used in cache, with a bottom-up and top-down approach for the implementation of these trees.

Algorithm Summary

The splay tree is a self-adjusting binary search tree data structure, meaning that the tree is adjusted dynamically based on the accessed or inserted elements. It automatically reorganizes itself so that the more something is accessed or inserted, the closer they are to the root. To be able to accomplish this functionality, we must perform various sequences of rotations on the tree, called splaying. Splaying essentially restructures the tree by making the most recently accessed or insert elements the new roots and moving the remaining nodes closer to the root. We must have insertion, deletion, search and rotation functions to create a functioning splay tree. In our case, we plan on implementing a splay tree to index datasets of varying sizes to assess the runtime and effectiveness of splay trees.

Function I/O

The dataset will be put into a tree and then through the functions we write it should be able to just make a splay tree. The functions we intend to implement include insertion, deletion, search, and a rotation. This will be accompanied with any helper functions we deem necessary. Insertion on a splay tree functions similarly to that of a Binary Search Tree where it must first be inserted in an ascending order into the tree. From there the node has a series of rotations applied to it to make it the new root of the tree. Deletion requires locating an element using regular BST deletion. If the element does not have children, then we can remove it. If it has 1 child, we need to promote that child to the element's position in the tree. If it has 2 children, we need to find the element's successor, swap its key with the element to be deleted, and delete the successor. Search on a splay tree starts with a regular BST search. If the element is found, apply rotations to bring it to the root; if the element is not found, apply rotations to the last visited node in the search, which becomes the new root. The 2 main rotations are zig and zig-zig rotations. The zig rotation brings the current node to the root. The zig-zig rotation balances the tree after there have been multiple accesses to elements in the same subtree.
[add how our tests prove correctness]

Data Description

The dataset we are using is a COVID dataset, and we are importing it as a csv file. Therefore, it is already comma separated, and we do not need any additional code to process the data.

Link to the github repo: https://github.com/snehac2003/CS225-splaytrees