Shaun Ryan

Professor Andujar

CS-300: DSA: Analysis and Design

10/10/2021

Data Structure Analysis

For this project, three different data structures were implemented and evaluated as solutions to the same problem. This was taking a list of college courses as a comma separated values (CSV) file, reading it, loading it into the respective data structure and checking it for validity. The vector was the first and most straightforward of the three. The input file was evaluated line by line (course by course), and each line was broken down to the individual elements of the course (number, name, prerequisites). These elements were stored in a Course structure and pushed to a vector.

The hash table was written similarly, however instead of simply pushing the node to the end of a vector, it was hashed by the course number and inserted into a hash table. This has the positive effect of making the courses quick and computationally inexpensive to find. However, it adds complexity and for the setup and insertion of nodes. The performance of hash tables is negatively affected by collisions, where it attempts to add a node to a bucket that is already populated. In this scenario, the table can be structured so each bucket contains a linked list of values, or an alternative bucket is found for the node according to its probing sequence. For this project I opted for the latter solution, but each has advantages and disadvantages. While conducting a runtime analysis, the worst-case scenario assumes that collisions happen for each node being added to the hash table. This negatively affects the runtime, leaving it at O(n^2), as opposed to a best case of O(n) where no collisions happened.

The tree data type was similar to the hash table as far as complexity and efficiency. The first course read was placed in a node selected as the root course with pointers to two child courses. As courses were added to the tree, they would traverse left if they were less than the node being examined or right if greater, until being added as a leaf to the last node examined. This data structure, while being harder to visualize than the hash table, was easier to implement. The runtime analysis was very similar to the hash table, having a slightly higher total cost and a worst-case runtime of O(n^2). This occurred if each new node added would have to traverse the entire tree before being added. This scenario is actually not that unlikely, as it would occur if the courses being added to the tree were already in alphanumeric order. The efficiency of the tree data structure seems heavily dependent on the arbitrary choice of the root node and how ordered the data being fed into it is. A technique to negate this could be hashing the course number prior to adding it to the tree in an attempt to get a more normal distribution. This would add a layer of complexity for searching for nodes later.

At the onset of this project, I wrongly assumed that the tree and hash tables would be much more efficient than the vector, but the opposite proved true. This is largely due to the fact that the creation and loading of the data structures were evaluated (where the tree and hash table are inherently more complicated). If a runtime analysis were conducted for searching for values, I am confident that the vector would fare the worst. For ABCU’s computer science department, I would recommend a hash tree data structure. It would be the most efficient for searching for classes, which would be a far more common operation than loading new classes and thus more crucial to prioritize runtime. The removal of nodes (courses) is also unlikely to happen frequently, which can be computationally expensive as a tree data structure would have to be refactored recursively. Given the number of courses a single department of a university would offer, either a hash table or a tree would be adequate. They are similar in terms of performance and are both relatively easy to scale. While the easy implementation and low computational cost of building a vector may be appealing, having a data structure that is far more likely to be read from than written to would make it a poor choice overall.