Project One – Analysis and Recommendation

Chris Richards

Christopher.richards4@snhu.edu

After writing the pseudocode and designing the data structures for the advisor’s requirements, there are some items of note that need to be discussed. For each of the data structures considered regarding storing class information, each of the structures needed to contain a vector for storing prerequisites for each class. This caused added complexity in both time and space requirements for each of the data structures considered. Since this was needed for each of the data structures in question, there should be no consequence in terms of recommendation for the application.

When analyzing a vector to store class data, there a few advantages and disadvantages noted. When searching for items in the vector, the use of the binary search algorithm made the time complexity for finding a class within the vector a factor of O(logn). Insertion into the vector is also vary fast, as the initial insertion can happen with a push\_back operation with a time complexity of O(1). While inserting and searching are within an acceptable time complexity range, sorting is not as strong as the implementation using a binary search tree. The time complexity using the quicksort algorithm on a vector came to be O(nlogn).

When analyzing the hash table data structure for this application, again there were some advantages and disadvantages noted. When using a hash table, the analysis indicates that the insertion of a class into the table has a complexity of O(N2). This is ultimately due to the need to split each line from the CSV file into a vector, as well as parsing each prerequisite. In reality, the time complexity for performing the actually insert on the hash table is more closely constant, or O(1). When retrieving classes from the hash table, since there aren’t any duplicate keys, the worst-case complexity for retrieving a class is O(n). As far as sorting the hash table, by definition, they are unsorted data structures. To allow for sorted traversal of the hash table, the application would need to be expanded to possibly use a vector to cache the hash tables keys. This vector would be sorted and used as a lookup reference for accessing the keys in the hash table. This adds increased space complexity with not much benefit.

Lastly, the binary search tree was analyzed for use in the application. Due to the need to include a vector for prerequisites, the binary search tree had the same limitations that each of the other data structures had in terms of traversing the data structure. When inserting the data into the binary tree, the time complexity calculated came to O(logn). Searching for a class in the binary tree also came to be O(logn). In fact, almost every operation performed on the binary search tree had a complexity of O(logn), with exception of verifying the course data. The time complexity O(n2logn). This was still significantly better than the hash table’s time complexity for the same operation at a staggering O(n3). One last advantage of the binary search tree is that the insertion algorithm used in a tree naturally causes the tree to be sorted when it is populated. The largest disadvantage to using a binary search tree will be the increased complexity in terms of implementation.

After these analyses, it is my recommendation that the application should be implemented using a binary search tree as the data structure of choice when storing class data. Due to the low time complexities involved in searching, inserting and sorting nodes, it makes an ideal candidate for this application.