Matthew Bramer

February 18, 2022

CS-260 Data Structures and Algorithms

7-1 Final Project: Reflection Paper

**Data Structures**

Data structures and algorithms are essential tools for anyone entering the field of computer science. These tools consist of specific patterns that, when applied correctly, solve problems. As a professional working in the field of computer science, you must be aware of the various data structures and algorithms that are available, and more importantly, when to apply them. Being familiar with different data structure and algorithm options allows you to solve problems efficiently, because you will not have to re-create a solution repeatedly.

**Vectors**

A vector is an ordered list of items of a given data type. Each item in a vector is called an element. Vectors are safer because the access v.at(i) is checked during execution to ensure the index is within the vector's valid range. An array access a[i] involves no such check. Such checking is important.

**Hash Tables**

A hash table is a data structure that stores unordered items by mapping (or hashing) each item to a location in an array (or vector). Ex: Given an array with indices 0..9 to store integers from 0..500, the modulo (remainder) operator can be used to map 25 to index 5 (25 % 10 = 5), and 149 to index 9 (149 % 10 = 9). A hash table's main advantage is that searching (or inserting / removing) an item may require only O(1), in contrast to O(N) for searching a list or to O(log N) for binary search.

In a hash table, an item's key is the value used to map to an index. For all items that might possibly be stored in the hash table, every key is ideally unique, so that the hash table's algorithms can search for a specific item by that key. Each hash table array element is called a bucket. A hash function computes a bucket index from the item's key. A common hash function uses the modulo operator %, which computes the integer remainder when dividing two numbers.

**Tree Structures**

In a list, each node has up to one successor. In a binary tree, each node has up to two children, known as a *left child* and a *right child*. "Binary" means two, referring to the two children. Leaf: A tree node with no children. Internal node: A node with at least one child. Parent: A node with a child is said to be that child's parent. A node's ancestors include the node's parent, the parent's parent, etc., up to the tree's root. Root: The one tree node with no parent (the "top" node).

**Algorithms**

**Search**

Given a key, a search algorithm returns the first node whose data matches that key or returns 0 if a matching node was not found. A simple linked list search algorithm checks the current node (initially the list's head node), returning that node if a match, else pointing the current node to the next node and repeating. If the pointer to current node is 0, the algorithm returns 0 (matching node was not found).

**Sort**

Sorting is the process of converting a list of elements into ascending (or descending) order. For example, given a list of numbers {17 3 44 6 9}, the list after sorting is {3 6 9 17 44}. You may have carried out sorting when arranging papers in alphabetical order or arranging envelopes to have ascending zip codes.

The challenge of sorting is that a program can't "see" the entire list to know where to move an element. Instead, a program is limited to simpler steps, typically observing or swapping just two elements at a time. So, sorting just by swapping values is an important part of sorting algorithms

**Hash/Chaining**

Chaining handles hash table collisions by using a list for each bucket, where each list may store multiple items that map to the same bucket. The insert operation first uses the item's key to determine the bucket, and then inserts the item in that bucket's list. Searching also first determines the bucket, and then searches the bucket's list. Likewise, for removes.

**Student’s Choice**

My favorite program that used both an algorithm and a data structure, was the Vector Sorting program in Module Four! Specifically utilizing the Quick Sort Algorithm and Selection Sort Algorithm. I think, obviously, the Quick Sort Algorithm would be used more often as efficiency is best in the workplace. However, when quality is preferred over speed, in which I believe the Selection Sort Algorithm would work best for that. I don’t believe there are many strengths for the pair solving a problem.

As mentioned, I feel, the Quick Sort Algorithm would be used more. I’ll have to see when I work in the development world. Would these two separate algorithms get used much together, though? Or would the focus be more on one algorithm than the other.

For my code to be modular, I feel the best use of code is built for the future. What I mean by that is, for scalability. For instance, the Quick Sort Algorithm is infinitely scalable. The Quick Sort Algorithm is undoubtably reusable in all situations, only having to edit the main variables.

**Conclusion**

Data structures are vitally important and will be used in my professional life. Developing best practices earlier will be key for continuing to improve in programming ability. Again, I cannot stress the functions and use-cases for the Quick Sort Algorithm in the workplace.

Lastly, I found the lessons learned from this course stem from learning how to learn. Because most programming languages have similar structures, it’s just the variability that is the key to understand for each language. There will always be a quick sort algorithm, but it might be different languages that it can be written in, for different functions. I will try my best to use all data structures and algorithms that I’ve learned in this class in my personal and professional life.