**The Concept of Data Structures**

Brittany Kirkham

Olympic College, Spring 2023

IS 390: Reading & Research

Prof. R. Becker

June 10, 2023

As future project managers in the Information Technology field, we are learning the fundamentals of team administration and the applications of technology. Our BASIS program is geared towards students transitioning into IT from unrelated professions. As such, our program seeks to instill a broad overview of technology features rather than going in-depth. For the most part this approach will benefit us in our future careers, however there are certain concepts that I believe would add to our utility if we were to understand their foundations. For this reason, I have chosen to explore the concept of Data Structures.

**Introduction**

A basic function of computers is to allow us to create, edit, store, access, process, and transfer data. This ability was made possible by the correlation of electrical pulses to the 0’s and 1’s of the binary system. For the sake of this research paper, I will assume the reader understands what the binary system is and how it translates the 0 and 1 bits into words, numbers, commands, etc.

A byte of data that reads: 01000100 could represent the decimal number 68, the ASCII symbol D, or could be a small part of a larger grouping of bits that serves to launch a command or process. So, the question arises as to how a computer can read a file full of 0’s and 1’s and translate the same data into the original input. To answer this dilemma many universal rules, setpoints, and concepts have been created.

Data Types are used to define the attributes of a set of data and to tell the computer how to interpret them (Franceschetti, 2016). It is important to note that data types don’t include data, they just define it. For instance, when a programmer wants to store the number 68, they will write it as:

int storedValue = 68

The int in this example is the data type, telling a computer that the following bits will be an integer and should be read as such. An additional process that takes place when a computer is given an integer to store or read pertains to the universal rule of attributing 32 bits to an int variable. This rule allows the computer to understand when to stop trying to read the bits as a decimal number. In a large block of bytes, the computer knows the data type is the starting point and once it finishes reading the designated number of bits anything afterward is a new command or data type prompt.

The issue of being able to read bits the same way every time is fundamentally addressed with data types and universal setpoints, however it now causes another large concern: memory size. A decimal value of 68 can be represented in binary as a single byte, but for it to be executable we must include extra bits to represent the data type, the variable name, and the 3 bytes of 0’s that are needed to satisfy the 32-bit integer rule.

When only data types are used every single integer, character, float, and Boolean will have to have its own data type, its own name, and its own extended designated bits. As you can imagine, a simple list of names and numbers can grow the size of its file exponentially. To mitigate this, developers came up with ways to better organize the data by combining data types with data structures (Franceschetti, 2016).

**The Benefits of Data Structures**

When your desk gets piled up with papers, notes, books, and pens it becomes much harder to find what you’re searching for. There might be duplicate notes, outdated books you’ll never need again, or maybe some useless snack wrappers that got buried and forgotten. To make your desk usable again, you need to organize it. By putting all the loose papers together in a folder, by stacking all the books together on the corner, and by disposing of all the trash you can cut down on the clutter and give yourself more *space* to work with. Organizing can also allow you to use your time more *efficiently* when looking for something specific. This same concept applies to the memory allocations of data.

Data structures work in organizing data by correlating it together (Keogh & Davidson, 2004). When a programmer has two hundred integers they need to store for later reference, they can correlate them together and store them in a single data structure. This structure will tell the computer all 200 numbers are integers and can be referenced by the group’s name and the integer’s position in that group. By defining one data type and one name for the entire set of data, the file size can be greatly reduced.

An additional benefit of utilizing data structures comes from the processing efficiency of the data. When trying to find the average of two hundred integers without a data structure, a programmer would have to “point” at every variable by name in the code (Keogh & Davidson, 2004). For example:

sum = storedValue1 + storedValue2 + storedValue3 + …;

This would continue until the variable names for all 200 integers were listed. However, with data structures the code for this process can be simplified to:

sum = 0;

for (int i = 0; i < 200; i++)

sum = sum + storedValue[i];

Although this might look a little confusing, it is a far simpler and smaller code than the original option.

Data structures are also used to establish correlations between the data sets which the computer can understand without the code specifically defining them (Keogh & Davidson, 2004). Let’s say we wanted to store 5 student’s names and their corresponding grades on the final. Traditionally, every student would have 2 variables defined for them, one for their name and one for their grade. However, the computer doesn’t understand that when student3 and grade3 were created, they were intended to be connected. For the program to understand that these two variables need to be correlated, the programmer will have to specify it. However, if a data structure is used to store the names and grades together, they can all be referenced at the same time without the added steps of establishing those connections.

**Types and Applications of Data Structures**

When organizing your desk there are multiple ways to do it. The different styles could end up impacting the usefulness of the desk or the access you have to your materials. Similarly, data structures have multiple options for the correlations they make and the ways they store your data. To better understand the different possibilities, we will look at how some common data structures are classified and applied.



(GeeksforGeeks, 2023)

**Linear Data Structures**

A linear data structure organizes its data in a linear manner. This means the data points are organized sequentially or in a fashion which connects them to their direct neighbors (GeeksforGeeks, 2023). This format allows for all the data to be read in one run. Linear data structures are further broken down into static and dynamic categories.

***Static Data Structures***

Static data structures are a type of linear structure which requires its size to be allocated at its creation (GeeksforGeeks, 2023). These structures are most useful when the *amount* of data doesn’t change. Take for example a soccer team’s starting roster; every position on the field must be filled. The names of the players filling those positions might change, but there will always be the same number of names on the list.

**Arrays.** The most common static data structure is the array. Don’t be misled, arrays can be made into different sizes by several different methods. However, it’s important to note that these methods only give the appearance of altered sizes while maintaining the characteristics of a static data structure (McDowell, 2020).

The first method is to write a code which checks how much data must be stored prior to each instance of creating an array (McDowell, 2020). This method is only applicable when the array is being generated in an embedded function. It relies on where the array is created and when the function it’s embedded in is called. For instance, having an array be created with data from a database every time a webpage is pulled up, like we did for the portfolio website.

The second method is to have a larger than necessary array which is filled with null values for every unused position (McDowell, 2020). By adding a command to not display nulls, this method allows for some adaptability in an array. This method can display a varying number of array sizes, but it still has a set size. This means it runs the risk of the size being too small or too big. If the size is too small, some of the data could be left out or an error could be thrown which breaks the whole function. If the size is too large, the array will still have to store all those null values and the program will still have to process them. This possibility means the array would need more space and would be less efficient.

***Dynamic Data Structures***

Dynamic data structures are the opposite of static data structures, in that they can adapt their size to fit the data they store (GeeksforGeeks, 2023). This adaptability makes them ideal for database storage configurations since their size can be reduced or expanded as needed.

**ArrayLists.** ArrayLists have an array-like structure with dynamic resizing. These structures act in the same way and maintain the same data correlations as an array would, with the only difference being the changing size (McDowell, 2020). A common implementation of these lists has the array size double or increase by a set amount once the array is full.

**Stacks.** Another dynamic data structure is a stack. This structure uses a “last-in, first-out” methodology, like a stack of plates (McDowell, 2020). The most recent data added to the stack will be at the top, meaning it will be the first to be read or removed. Data can only be added or removed from the top of the stack. The third data set can’t be accessed without going through the previous data sets first.

An example of a stack you might recognize would be the “Back” and “Forward” buttons on your web browser. This stack stores your browsing history for potential recall if you need it. However, to get to a specific previous web page you might need to click the button multiple times depending on how many sites you navigated.

**Queues.** Queues are similar in characteristics to stacks, with the exception of being “first-in, first-out” (McDowell, 2020). This methodology means data is added to the end of the line, while data is taken from the front. Queues are very common, both in computing and in our daily lives. Any time a line forms for purchasing something, or a waiting list forms to access a service, a queue is used.

**Linked Lists.** Linked lists are another form of a list data structure. In this iteration, the list must be navigated in order from the current position to the destination (McDowell, 2020). Like an array, each data point is connected to its direct forward neighbor in a singly linked list, or both its neighbors in a doubly linked list. This means the list’s data is not constantly available. If the program calls for the data in the third position of the list, it can’t just be provided. Instead, the data at positions one and two must be navigated before the third can be accessed.

Linked lists are commonly implemented in tandem with other data structures. The previous two data structures are limited in that they require the use of an additional data structure for functionality, like an array or linked list. This nesting structure is due to the stack and queue design, which focuses solely on the positioning and the next data set it needs to provide (McDowell, 2020).

Imagine a line of people waiting at a meat counter. The queue data structure would be the little number slips that everyone takes and the digital display that shows which number they’re helping. However, the queue doesn’t care about how many numbers are taken or who takes them. Instead, the data pertaining to the waiting list is stored in an array or linked list. In this way, once the queue calls for number 7, the attached data structure can provide the data associated with that position.

**Non-Linear Data Structures**

As the name suggests, non-linear data structures store the data in a non-linear fashion. This means the data is not stored in sequential order, which means the data can’t be read in a single pass (GeeksforGeeks, 2023). Added steps and code are needed to fully utilize these structures, which has classified them as more advanced structures. To understand these structures, it is better to visualize them.

**Trees.** Tree data structures allow for correlations between data sets that are not necessarily directly next to each other (McDowell, 2020). If this structure were to be drawn out for visual reference, it would look like the diagram of data structures above. One “parent” data node can be connected to several “children” data nodes which gives the impression of a tree with many branches. In computing, tree data structures will often be limited to 2 “children” nodes. This feature is due to the direct correlation of nodes to binary and is referred to as binary trees.

**Graphs.** Graphs are another form of a non-linear data structure. Many people confuse this structure with being linear when trying to visualize it, since we typically think of mathematical graphs. However, graphs in the programming sense look more like tree data structures with extra connections (McDowell, 2020). In fact, tree data structures are a type of graph data structure. When picturing a graph data structure, imagine a few nodes of data located next to each other with arrows connecting them together in different ways. Some nodes might be connected to only one node, while others connect to several. We used this graph structure in project management when determining the critical path of a project’s timeline.

**Conclusion**

Data types and universal setpoints are used to translate blocks of bits into readable data. Data structures, on the other hand, hold and organize data into different configurations and with different correlations. The use of data structures limits data storage space, as well as the lengths of commands and functions. Data structures also allow for adaptable processing and faster accessing of the data.

The different data structures are separated into several categories based on their characteristics. Each data structure type has its own strengths and setbacks that must be taken into consideration when working with them. The structure types described in this paper are far from exhaustive, however they include some of the more useful data structures that we will most likely come across in our future.

When each data structure was introduced, examples were given for their applications; however, these examples are very simple designs. It is very common for data structures to be nested or mapped to each other. By combining the different data structure types, a limited data configuration can become more versatile and useful.

More advanced applications of data structures include the use of hash mapped to an array index which holds a linked list to mitigate collisions (Brass, 2008). You might come across complex configurations like this, but with the basic concept of the different data structure’s strengths and weaknesses you should have a better starting point in your path towards understanding them.

**References**

Brass, P. (2008). *Advanced data structures*. Cambridge University Press.

Franceschetti, D. R. (2016). *Principles of computer science*. Salem Press, a division of EBSCO

Information Services, Inc.

GeeksforGeeks. (2023, May 3). *Data structures*. GeeksforGeeks. Retrieved May 5, 2023, from

https://www.geeksforgeeks.org/data-structures/

Keogh, J. E., & Davidson, K. (2004). *Data structures demystified*. McGraw-Hill/Osborne.

McDowell, G. L. (2020). *Cracking the coding interview* (6 ed.). Palo Alto: CareerCup, LLC.

Purdue Online Writing Lab. (n.d.). *APA formatting and style guide* (7 ed.). Purdue Online

Writing Lab. [https://owl.purdue.edu/owl/research\_and\_citation/apa\_style/apa\_ formatting\_and\_style\_guide/index.html](https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_%20formatting_and_style_guide/index.html)