Zachary Meisner

Final Project

CS 260 Data Structures and Algorithms

10/10/21

I. Data Structures: In the first section of your report, indicate which programs in your portfolio best exemplify your knowledge and skills related to data structures. For each example that you provide, defend your program in terms of how effectively the data is organized, using specific evidence from the problem. Specifically, you must discuss each of the following data structure types:

  A. Vectors

  B. Hash tables

  C. Tree structures

A.

In module 4 we are asked to use Vectors to implement several types of algorithms in the program. It is important to note that we specifically are trying to display distinct types of search algorithms, such as selection sort and quicksort which will be discussed later. For this program Vectors really were the star implementation here, as we are trying to show two different sort algorithms, vectors are generally an excellent choice for these instances due to the way they interact with one another, the user, and the program. To break it down, vectors are a type of internalized list of data points, added by the user of the program, and even by the developer of the program in the case they are making some type of system to be used by a company. One example of this is if we had been creating a type of social media platform where we had needed infrastructure to build individual profiles, perhaps so that every time someone wanted to add a friend, the system would add said person to the vector of friends that this person had, and in addition to that, running algorithms against each vector in order to compare them to find ratios of likability between different types of people based off of their internal data that could distinguish whether they would be more or less likely to get along with somebody. In addition to this, we could also find out whether there are mutual friends if we were to compare each vector with one another which could make it easier to make connections amongst a greater network.

B.

Module 5 is the best example of hash tables, but what exactly is a hash table? The best way I think to explain the theoretical structure is using that of a honeycomb. Hash tables, also are one of my favorite structures to talk about due to their makeup, later we will explore the reason I love them so much and all the other things that you are able to do with hash tables. To explain the basis of the ideological structure, we are not only talking about a honeycomb, but also about multilinear ideological existence which we can deduce to explain as objective reality. In a program that is object oriented, we are drawing different programs into the main stream and output of our reality, or the desired output of the program, creating what we can think of as a spiderweb, or a network. The basis of this network is important to think about because this is what supports the entire structure of the network and is one of the big reasons why it can support the weight or traffic, and what helps us understand theoretically what scale is on a much larger level, and what goes into deciding what does or does not have scale. Essentially, I am talking about vertical factorial, or exponential growth over time. To tie this all together, if you were to think of each spot inside the honeycomb as a bucket with x amount of depth available, you would also be creating the basis of the structure in an object oriented program, but alternatively, we are talking about the backend of software, a database of sorts where we are able to manipulate data in many different ways due to the basic existence of the program itself, and the potential possibilities we are able to do with this data because of the way it is stored. There are so many other potential things we can use this type of data structure for, and often it is paired with the ideological multi perceptual reality, or cross dimensional reality which is the basis for simulation theory. The fact that we can draw likenesses from math and computer science and chart it into our real world is nothing short of amazing, as visually everything can be seen interwoven together just like a honey comb. A strong use for something like a hash table can relate back to that of visualization of a computer on a computer screen. The mathematical projection of light is based off many different layers stacked on top of each other, so if we were to imagine not one, or two, but three hash tables vertically or horizontally stacked on top of one another, and then a stream of information, or light pushed through each one of these buckets of information, we can start to implement color, and use the stream of data to exponentially grow the output of these basic information that are now all integrated in with one another. In addition to this, if we were then to think of the natural state of a rainbow, or an eyeball, we could understand why hash tables are so pertinent as a structure for practical use.

C.

Lastly, when it comes to the use of Trees, we have a more peculiar, or interesting example of why they are so important, as when it comes to Trees someone may be able to question the reason, they are so important when they are so alike linked lists when it comes to the way that their information is stored. The only issue with this is that although they are like linked lists, the structure and organization of the information is unique in the sense that the way it is programmed is balanced, and can cause issues in the case it is not balanced properly. What I mean by this is that we are looking for some type of equilibrium within the dataset to deduce certain answers. The basic understanding of this shows that we are dealing with a lower level of data and that we have cut through a lot of the basic information, at least when it comes to the implementation of what we are working. through in this problem specifically. Trees can also be used in lightweight ways too, for example if you are working with two separate profiles in a video game and you would like to compare scores of different tasks that each profile has done, Trees are the easiest way to do such a thing because you are able to have that balanced structure, while also being able to divide up answers from the data you are using. To get to the true optimization of this data structure, we must think in an algorithmic sense what this could be used for. In real life, we can think of trees having the structure like that of a family, for example it takes both a mother and father to birth a baby boy, and the file structure starts to show that of what we could consider to be a reverse pyramid or a downward triangle, but realistically I think what it is mimicking is the transfer or flow of information on a much larger scale when you dive further into considering the implementation of quarks and atoms. What makes this so incredible and interesting is that this is the natural state of things and the flow of information within this file structure simply just works because when it is integrated in with other types of information, we may need to process there is not that much you have to change within the correct usage of this structure, such as if we are going to have a computer compute a shape based off a mathematical algorithm. One that comes to mind is a shift and add algorithm used to show a machine how to make circles within a program based off beginning set parameters given by the user.

II. **Algorithms:** In the second section of your report, indicate which programs within your portfolio best exemplify your knowledge and skills related to algorithms. For each example that you provide, defend your program as one that is technically sound and can perform the required functions, using specific evidence. Specifically, you must discuss each of the following algorithm types:

A. **Search**
B. Sort
C. Hash/Chaining

A.

The search algorithm is one of the most basic, yet important algorithms that we have in our toolbox to use. The idea is simple right? Let us say that you dump a bunch of Legos out of a box, and then in front of you have a giant unsorted pile. The search function within any program will allow your computer to search through each Lego, as you were individually looking for specific indicators within set parameters of each Lego to find whatever it is you are looking for. This really helps set the basis of a foundation for other programs as well because if you do not have a good searching function within your program, not only is it frustrating to use, but it hurts all the other parts of your program as well because organization is pertinent to have within structure of any kind to implement the purpose of what you are looking to do in or with the software. Within module 3 we have a notable example of a basic search function run against a linked list, and we can see exactly what the algorithm does and why it works so efficiently. A linked list really is what it sounds like, a list of items that are glued together, but only on either side of the object. Whether you imagine this vertically, or horizontally it does not matter a whole lot if you are understanding the basic idea of what we are going with here, such as you can find commonality in that of a grocery list, just as much as you could if you took a deck of cards, and put them all next to each other face up in order across a table. Now what the search algorithm does within the linked list, is it takes the first key it is given, in this case it is the ID number of the bid, and individually looks at every single item within the lists that it has, searching for corresponding ID numbers within each part of the list. Once it is done, it outputs the answer of whether it does or does not have the corresponding ID number, telling you exactly where it is.

B.

To further explore searching, we can investigate other types of algorithms as well. As I had said, searching being one of the most basic functions of a program can branch into other functions as well, really integrating the entirety of the structure together which is one of the huge reasons that it is so important. One of the ways we can help the search function do this is by integrating specific data structures and algorithms that are great when it comes to sorting data and information. Whether this is inputted by the user or developer of the program, it is pertinent to make sure everything is organized in a way that is easy to understand. What this does is help the computer. Because of this the program is given mathematical properties in addition to the basic understanding of computation along the data structure, we can exponentially grow the abilities of the program and the computer to make the function the most efficient it can be. We can see this the best in both Module 4, and Module 6.

In Module 4 we are implementing a structure using vectors and running algorithms on the vectors to compute the data in the proper way. More specifically we are using Selection Sort, and Quick Sort. Starting with selection sort, the way we can think of this is if we first dump a pile of cards on the ground and then line them up in any order on a table in front of us. Noticing that this is not the correct way the cards are supposed to be organized, we start at the beginning, comparing the first value to the card on its right, if the value of this card is lesser than the original value we are comparing to, then we move it to the left, and continue the process that we originally had prior to swap values and find the correct order. After a certain amount of time, we eventually will have the entire deck of cards from the lowest value to the highest value. On the other hand, we have Quick Sort which is similar but can be thought of on a greater scale, but with a lesser number of values. What we first do, is if we had the cards in front of us unorganized and in any order, we would choose a pivot point in the cards. What this means is that we would look at all the cards, and decide that one card is where we start and then we would go from there, and it can be any card within any type of specified parameters, meaning that we can segment the card list if we want to try and tackle smaller portions opposed to the entirety of the list. Due to the ability of this we also consider this to be a type of partition algorithm since we often use it to split up specified portions of values. The ability to choose your pivot varies as well along with these decisions following the partitions, maybe you notice that it would be easier to use the median number in order to sort the cards, or maybe the pivot would be easier if you started with the first card, or even the very end due to the way that you notice your data maybe presorted, for example if you notice that there are a lot of larger valued cards in the beginning, perhaps you would want to start at the beginning due to there being less lower value cards, which would take significantly less time if you were to partition through with the quick algorithm from one specific point of the dataset. To further understand exactly what is happening, when we pick this pivot, in addition to the original decision of where we want to have the pivot within the given dataset, the dataset then partitions again around the pivot, comparing values and taking larger values, putting them on the right side of the pivot, and lesser values on the left.

To finish with Module 6, we are using something that implements a data structure known as a Binary Tree. What this means is that like the past example with the pivot, the tree has an original root in which the entirety of the network of data is connected to one another, and then the data is divvied up onto either side in a specific way due to the content of the data, making sure that the tree is weighted so that we can properly compute through the values, and make sure that we are not leaving any holes within the tree and that everything is even. After this, we then can run traversals on the trees in different specified orders to understand the data and organize it in the way that it specifically is supposed to be organized. A traversal is a specified order in which the computer is supposed to visit each node based off the structure of the tree. So the output you get back is based off of what you specifically are telling the computer to do, with the added addition of having your data segmented based off of differing data sets, so even in the case you know on the right side of your tree, a number may belong in a specific place due to the order on the left side, it will still be considered as separate, allowing you to do a lot of different and cool things with your data on a lower level of utilization within a computer science spectrum, for example when it comes to pointing towards different parts of your computer that you want to use when beginning the creation of an operating system, and telling your computer where it needs to put the basic amount of electricity that it has to compute with, in order to create a basic structure in order to help higher level elements compute the most efficiently.

C.

Continuing to Hash Tables and Chaining, like I had said prior in the paper, if we were to think of a Hash Table the image, we can use to understand what is going on is like that of a honeycomb. Now, inside of each part of the honeycomb is a bucket, in which you can dump whatever you would like into each bucket without it interacting with one another specifically. The way we divvy up what type of substance should go in each bucket though is with a type of key, and based off the way these keys are put in specific spots is what decides the order of the data and where it goes. The issue that arises here when we do this is if we have two similar substances that we want to put into each bucket, they may start with the same letter in the beginning of their name, such as water, and watermelon lemonade. Or another example, is if we have two types of water, that is specifically from different time portions in a day, or even various places in the world. Due to the similarity of their names, they may be assigned the same key, meaning that they both end up being put in the same bucket which creates a problem for us called a collision. This is a problem because every part of the hash table is only supposed to have a single slot for each type of data, and this could cause a lot of other issues due to the possibility that the entirety of the order for each of the data in the dataset to mess up due to them being assigned to the incorrect spot. The way that we can account for this type of problem is with an algorithm known as Chaining, which is specifically meant for Hashing, as no other data structure really has this type of problem. This is interesting because it uses linked lists, and evolves the structure, letting it go from 2D to 3D. What I mean by this, is if you were to think of a Hash Table, like I had said prior we can think of the shape of a honey comb. Now if you were to imagine each row of each hole in the honey comb, and partition each part vertically from the top to the bottom, we would just have linked lists stacked right on top of each other. In addition to this, if you were to turn the honey comb to the side, and imagine as if you could see inside each bucket looking from the different angle, you could imagine that each point of data within each bucket had an individual spot, but what chaining does, is in the case there happen to be any collisions due to the similarity of data or keys, we create linked lists inside each of the buckets as well, essentially squaring the amount of linked lists that we have allowing the data structure to grow exponentially with the software in real time along the amount of computing power that we are using to match or exceed the volatility of communication. What this specifically becomes good for, is the implementation of several types of virtual reality. Not only are we able to understand and conceive the amount of light and color within specified data sets using these hash tables, but we are able to communicate with our eyes and brain, creating what is perceived to be multi-dimensional space within our own virtual instances specified by our brains using similar techniques. The flow of this constant really is incredibly impressive and the fact that we even have been able to understand these ideas let alone conceive how our brain works is amazing. The amount of information we may learn from ourselves and the way that are brains work, or even how we can begin to understand or conceive consciousness I believe lies in the world of understanding chaining, hash maps, and hash tables, amongst multi linear dimensional space using these types of algorithms.

III. **Student's Choice**: Select your best or favorite program that uses both an algorithm and a data structure.

D. Defend the **overall effectiveness** of this computer program. To what extent did you implement the algorithm(s) and data structure(s) to create a program that works? Support your claims with specific evidence from your work.

E. Defend your **programming abilities** as they are illustrated in this program:

1. To what extent is your code **modularly** composed? Support your claims with specific evidence from your work.
2. To what extent is your code **reusable**? Support your claims with specific evidence from your work.
3. To what extent are your **annotations** clear in explaining your choices and underlying reasoning? Specifically, how are your choices logically based on the needs of the case? Support your claims with specific evidence from your work.

My favorite program that uses both Linked Lists, and Searching is Module 3. The reason that this is my favorite program is since any of the more complex programs and algorithms, like discussed before, would not be possible without Linked Lists, in addition to the implementation of a great search algorithm that allows us to orient and compute diverse types of data within a variety of different structures, again, all thanks to linked lists. One example of a high-level implementation of this comes in handy when making distinct types of video games.

As we previously have discussed with virtual reality, we can recreate the structure of differing planes of existence based off where we program different decibels of light on the screen to be specific within the mathematical parameters we set. When we deduce this idea down to the base level of its understanding, what we realistically are conceiving is that of the transference of different decibels of data, or the transference of information at a type of light speed, only recreated and re transmitted on a computer, copying what is happening in real life. The study of our real world is incredibly important as we need to be able to understand exactly what is going on to properly recreate it on a computer so we can understand more about the secrets of our universe, such as what happens when we begin to evolve different socioeconomic or hierarchical structures on a computer, or even the idea that decentralized power may have better uses than centralized power does. Though we will not explore those ideas specifically, we really owe computers for giving us the opportunity to explore these different potential avenue of studies as we would not be able to do any of it without them.

Truly what I think is important to understand is that among the transference of data that is implemented by the algorithm, we are recreating algorithms amongst the transference of the data, which is to an extent mimicking what we are doing on a lower level, at a higher level so we can understand it visually with our eyes due to the constant, or even non constant of patterns that are brains are used to conceiving on a day-to-day basis. Within the work, we can see specific implementations of similar patterns when it comes to sorting and divvying through the information, as we can see that it follows the same patterns that most shapes do within Euclidian geometry as we are seeing Euclidian Super Symmetry and geometry on much higher levels within data structures that take shape into higher level geometrical prisms that are essentially showing us what high level systems look like, for example the social networks amongst Facebook, and the patterns of weighted graphs that comes from the connections and linked lists for each individual person, what they like, and who they are close to.

This brings us to the next point of modularity, as we can segment each part of our program to implement different algorithms to expand our structures and algorithms to make it easier to understand what and why certain things are happening within real time. For example, if we can use linked lists to understand why certain people like certain things, due to the likelihood they have for exposure based off geological decisions, or even sociological brands that are biasing them based off their childhood and what they may or may not have been more likely to see as a child. The fact that we can segment so many distinct parts of our program, especially within linked lists, shows the exponentiality of potential we have at our fingertips. The reusability here due to the modularity really is again, exponential, because we can go back in time to track the usage of these structures and algorithms way before medieval times. For example, if you were a peasant farmer the natural state of evolution shows and proves that if you have allotted space within your given land then it makes more sense to line up vegetables and pair them together to make it easier on you, so when you go to harvest them you can know exactly when different portions of your farm are going to be ready for harvest, which parts not to harvest, and where everything is, essentially showing the same ideological structures of linked lists when it comes to the implementation of systems that work together that follow higher level systems designated by the humans in charge to make their schedules more efficient.

The reusability can be tracked repeatedly in different instantaneous expressions throughout time since we are realistically traversing through what we can consider to be multiple realities of patterns and loops that copy until the end of the weighted graph we are creating to the point where we inadvertently change the structure to evolve into something different. The best way that I like to think about what we are doing within the field we are in, is that of modern-day Data Farmers. We can even see this in real life and how it is that people mine cryptocurrency using the data and power allotted on their computers. Within our work we can see exactly what I am referencing when we explore Hash Tables and Chaining, as I previously explained the use and importance of linked lists which shows the reusability in a different setting as well as the evolution of the several types of structures based off other structures. Lastly, to talk about the clarity of annotations when it comes to choices on any given project, I find it most important to talk about the understanding of what we call the scope of a project. When making pseudocode to pitch to someone, we must make sure that the code is understandable, and that when we are choosing what features to implement in the program, we are answering every issue, or foreseeable issue within the code based off the set of basic parameters given to us by our boss or the client. In the case that this can be overcomplicated, or cluttered, it loses its finesse and point. This clutter can cause extra confusion in the long term in the case the program or part of the program you are working on is implemented in with another feature of the program, or worked on by a different person.

In the case our annotations and decisions are not clear within the scope of the project, it is easy to get off track, amplifying the potential bugs within a program, again especially if you are not considering the further implementation of your features with others. For example, if you are choosing to create a program that does not implement chaining within your Hash Table, but then your program is chosen to integrate in with the rest of a software that also uses Hash Tables but DOES use Chaining, it is very possible that a mistake can be made, or we can have cross fire amongst the data set and allotted tables leading to security holes or giant crashes of the program itself due to something going where it is not supposed to go because what you originally made was not clear. To show evidence from my work, we can see that the number of features available are reduced to necessities, Append, Prepend, Insert After, Print List, Remove, and Search. Now, it was possible that we could add more features, and make something much cooler, or even use the implementation of our linked list in the attempt to make some type of hash table that uses chaining, but for the specific project we are not doing that, and it is not needed. Although in the long term of the scope, it would be smarter to do that with scale, we are just starting small, and need to create a base system to jump off so we can make sure that the basic systems are implemented before the company decides to reach for the stars.

IV. **Conclusions:** In the final section of your report, reflect on the broader principles and takeaways from this course.

2. First, assess the role and importance of **data structures** in developing computer programs. How does the type of data structure impact or constrain the development of an effective computer program? Support your claims using specific evidence from the course materials.

3. Second, assess the role and importance of **algorithms** in developing computer programs. How does the choice of algorithms impact or constrain the development of an effective and efficient computer program? Support your claims using specific evidence from the course materials.

4. Finally, articulate the **lessons learned** about solving processing problems. Specifically, how might you employ your knowledge of algorithms and data structures to solve a processing problem that is of interest to you in your personal or professional life? Illustrate your response with specific examples.

In Conclusion, when it comes to assessing the role and importance of data structures when developing computer programs, we can see now that the type of data structure changes the scope of what we are choosing to do and create. You would not for example, want to use a linked list to compute substantial amounts of data, as the algorithms implemented are not strong enough and it would end up taking too much time, or computing power, making it so we are not running as efficiently as possible within the given program. When it comes to giant database projects, we may want to use lower-level data structures such as Hash Tables or binary trees, as a linked list would constrain our project way too much due to the size, and the fact that we would have to sort the data in a specific way, instead of using traversals to order a tree which would prove to be much easier in the long term for the basis of a file or data system for a company.

The importance of algorithms goes hand and hand with the importance of data structures because you are not always able to implement the ability of every algorithm amongst every data structure that we choose, so we really must understand the data that we are looking to compute, and ask ourselves what it is we are looking to do with this data, and how we need to compare, contrast, or even integrate to find the conclusions we are looking for. For example, again, we cannot really use the implementations of chaining within a binary tree as that would not make a whole lot of sense. It would be more proper to use chaining within Hash Tables because that would allow us a lot more freedom within the spectrum of our problem set that we are trying to answer. It is important though to note, within a Hash Table, the Binary Tree is the king of presorted search algorithms, so it may take less time to use a Binary Tree within specified allotted data that we need to sort, and organize due to the natural structure and flow of information that we are given.

Finally, to bring all of this together, the lessons learned truly are emphasized when it comes to understanding how to think systemically, how to think ahead, and how to see the broader picture so we can account for everything within the given task that we have in front of us. We have set parameters to account for when it comes to any given project, and we must consider that often in the beginning of our project we are not able to account for all the future problems that we are going to have thrown at us. In addition to what I mentioned above, it is important to be able to understand exactly how we can pivot between different implementations of our data, and create structures and algorithms quickly so we can use the toolbox of what we have in the best way, and show the different outcomes available in the case something must change last second. The best way I can think of to employ the knowledge when it comes to real life, is understanding exactly what is going on within the systems that we are looking to recreate. Understanding and drawing parallels between the structure of what is going on, understanding where the base of the program would be, and what functions are needed for everything to compute properly, and even understanding how two functions may integrate into one function through a series of computational parameters, and what may change within the environment of the dataset after the computation. For example, if we were trying to make a program for chemical engineers, and we were going to understand how to take gas, and transmute the gas into a liquid, and transmute the liquid into a solid. We must account for the changing of all these parameters within the dataset and how the data is affecting everything else it interacts with at each stage, in addition to the possibility that there may be implementations of perceptual multistability at each of the stages due to our biological parameters that may fool or trick us into thinking one thing is happening, when it is an anomaly created by our dataset that may throw a monkey wrench into the entirety of our program.