Holly Beeson

Data Structures

Lab 2

October 23, 2018

For this lab, I used a 2-dimensional array to represent the adjacency matrix. This made sense because the matrix is organized in the same way as a 2D array. Also, it was important to be able to access a point in the matrix using its x and y coordinates, because this corresponded to the "to" and "from" nodes on the graph representation.

I decided that using classes was not necessary in this lab because the adjacency matrix stored all of the information that was needed to find all possible paths in the graph. In my program, "nodes" correspond to the points on the graph I am finding the paths between. The x and y locations in the adjacency matrix represent the nodes. I included a function called findPath that found every path between the current node and all other nodes. This function included the recursion, which was how each path was found. The stopping case was when the end node was reached, and the general case was call findPath on a smaller subset of the matrix. I then wrote a displayAllPaths function that displayed all of the paths between all possible nodes. This allowed me to see the output in the console as well as write to an output file. getPath is a function that returns the path from a source node to a node at integer i. I also had functions to read in the input file and make the matrix, as well as to write the matrices and results to an output file.

The program was efficient in terms of time because checking if two nodes had an edge between them is O(1). It was not as efficient in terms of space. Making the matrix is $O(n^2)$. Writing this program, I learned more about recursion. I think it is a really useful tool for making a larger problem into something more manageable. I still need a little bit more practice with making my brain think in recursive terms, though. Next time, I think it would be helpful for me to map out what my output should look like for a couple of the input cases. I ended up doing that for this lab, but it was after I had started coding. It would have been helpful to have the desired output from the beginning so that I could see where it was going wrong when I was debugging.

Doing this problem iteratively would not have been as efficient in terms of memory. Comparing every possible pair of nodes one at a time would take up a lot of space. Another issue is that we would have to know how many times to iterate at the beginning of the function, which is not feasible for this problem because our adjacency matrices can be different sizes. In terms of time, the efficiency is about the same for an iterative solution versus a recursive solution.