Audrey Long – Lab 2

This lab focused on delivering a program that reads in graphical nodes from an input file and finds all of the possible non looping paths between all possible pairs of nodes. Let's take a deep dive into each function of my program and provide justification for each data structure used. This program requires a lot of recursion, so methods were implemented to keep the code clean and easier to follow, these methods include:

- Findpath - This method takes in an adjacency matrix, a few arrays to store values of visited and parent nodes, and integers to represent the start node, end node, first node, and last node. The purpose of this method is to find all possible paths from a selected node.
  This function uses recursive calls to find multiple paths from the selected node, this strategy was used because the node has multiple paths available in order to find all of the paths that do not include a cycle, with such redundancy we can use recursion to simplify the code. An adjacency matrix was used to represent a vertex graph (g = v,e) which costs O(1) to add to the adjacency matrix where it costs O(v^2) to search. Arrays were used simply because adding values to an array in a specific order costs O(1) and searching the array only costs O(n).

- writeAdjMatrix - This method writes all the possible paths from an adjacency matrix to an output file. The inputs include the adjacency matrix we are reading from, and an output file to write the results. This was made into a method so we can easily read the stored values from the adjacency matrix in one function call.

- allPossiblePaths - This method shows all of the possible paths taken from a node. The input includes an adjacency matrix, and the output is a string that shows the possible paths from the given input. This function is very important and used throughout the entire program, it finds all of the possible paths from a selected node and stored that information in an adjacency matrix, the reasoning for putting it in a function was to recursively use it throughout the program.

- getPath - This is a recursive method to get the path from the selected node which it reads from the method allpossiblepaths and prints the results one at a time in a string buffer. This function is useful to grab one trace path from a selected node at a time, the reason why this was put into a function call was to reuse the call throughout the program since many different paths are contained in the input file.

- readAdjacMatrix - this file reads in an input file and populates an adjacency matrix, then calls the method allpossiblepaths to determine all of the paths from the selected node, and writes the results to an output file.

This program has taught me alot about graph theory, I had to take an algorithms course my undergrad, but never had to program any trees or graphs, i think this exercise has been very educational in implementing such graphs. I have learned a lot about the importance of recursion and how to properly implement recursion in a program which would be much more verbose than the current result and redundant.
I made my code modular with many functions to keep my program from being so redundant, also since the calls are used so often, it only made sense to make functions out of them. I also used the adjacency matrices as they were required in this lab, but also to store information from my program.
If i were to implement this program again i would probably try to make use of recursion in a different way, i also think i would refactor m allpossiblepaths and getpath function to be one in the same, for they both grab a path, it might make the code easier to follow.
During the creation of this program many exceptions were introduced in order to keep the input from the user from being incorrect, to throw an exception if the file path was incorrect, and general exceptions.