Lab3 Write-Up Abby Spangler

The two data structures I used in this project were a linked structure and a stack. My linked structure consisted of a hybrid implementation where I had the vertices stored in an array and each vertex in that array was linked to a linked list that held the edges adjacent to that vertex. I made a vertex class which contained information about each vertex such as its value and a pointer to its linked list. I made an edge class where each edge had a value, a pointer to the vertex it went to (not came from), and a pointer to the next edge. These pointers made it really easy to move from a vertex to its adjacent edge and then on to the vertex the edge led to. Since stacks are really useful for DFS, I re-used the same stack from Lab2 and I used it in the exact same fashion as I did in lab 2. My DFS method was also recursive as it was in lab 2, however, I had to re-write a good bit of the code to get it work with a linked structure instead of a 2D array. However, the main concepts remained the same. The base case is when you reach a vertex that has no adjacent or unvisited vertices. The run time for this algorithm is constant, O(V+E) where V represents the number of vertices and E represents the number of edges. However, recursion would use more memory than iteration because it would use a stack frame for each method call. An iterative algorithm would all happen in the same stack frame. If you are working with a very large graph, this recursive method would be a lot of method calls and you could potentially get a stack overflow. On the other hand, you could still utilize the stack data structure in the iterative method. An addition I made to my program was a 2D array for keeping track of where paths existed and didn't exist. It's similar to an adjacency matrix. If a path exists between two vertices, the value at the intersect of those two vertices is changed to a 1. At the end of finding all the paths, I was then able to see between which vertices no paths existed.

I don't find the linked implementation to be very appropriate for this application. First of all, it's way more time and energy intensive to implement than a 2d array especially when you can't use the pre-made classes. Additionally, in a situation like this you don't need to rearrange data so having pointers is not really an advantage. Also, you know ahead of time when you read in the data how many vertices there are so, you don't need the flexibility of a linked list having a dynamic amount of space. So, in my opinion, a 2d array is a far simpler and more efficient way of handling a problem like this. The upside to the linked structure is that you don't have to store the 0's from the adjacency matrix, you can just ignore them since only the 1's represent edges.

From this lab, I learned how complicated linked structures can be and how extremely important it is to know ahead of time which pointers you will need to make your structure functional for the problem you're trying to solve. Additionally, and most surprisingly I realized that once you think about and solve a problem recursively, it's difficult to go back to thinking about it iteratively. This was surprising to me because recursion is so difficult to grasp in the beginning whereas iteration is not.