Elba Chimilio
COP3530 – Section 7303

Project 3: Dijkstra's Algorithm Commentary

For my project, I implemented an adjacency list. The reason I choose an adjacency list is because it is memory-efficient, and it avoids out-of-bounds issues associated with arrays. Although an adjacency list is harder to implement, it has various benefits over an adjacency matrix. With an adjacency matrix, you must have information on the size needed to avoid wasting memory. In addition, it is not as effective in iterating through to find an item. An adjacency list is quick at iterating through but finding specific items may take longer than it would with an adjacency matrix. However, for Dijkstra's algorithm, it was more important to have a quick iteration through the graph to find the shortest path versus finding a specific vertex or edge.

Dijkstra's algorithm, represented through an adjacency list and with the implementation of min-heap, has a computational complexity of O(ElogV), where V is the number of vertices in the graph and E is number of edges in in the graph. I implemented a min-heap into my printDijsktra(int source) method. With the min-heap, I was able to reduce the O(V^2) computational complexity, where V is the number of vertices, of Dijkstra's algorithm, to O(ElogV). The min-heap served to check through each considered source vertex and find the shorter path. Once it found a shorter path than the previous, it would update the heap and keep running through until the shortest path was found. I learned how practical implementing different components such as vectors and min-heaps can be to help solve algorithms, however, it is not intuitive and does take some beforehand research into how these different components work with one another. If I had to do things over again, I would consider taking more time from the start to really think through why I would choose an adjacency list over the matrix since it seemed challenging for me to implement. While an adjacency matrix seems simpler to implement, I would fear having to try out several test cases to avoid getting incorrect results. Nevertheless, given the opportunity, I believe I would implement an adjacency list again since it seems like a more efficient method for this project.

- void insertVertex(int vertex) | **Computational Complexity:** O(1)
- void insertEdge(int from, int to, int weight) | **Computational Complexity:** O(n), where n is the number of vertices in the graph
- bool isEdge(int from, int to) | **Computational Complexity:** O(n), where n is the number of vertices in the graph
- int getWeight(int from, int to) | **Computational Complexity:** O(n), where n is the number of vertices in the graph
- vector<int> getAdjacent(int vertex) | **Computational Complexity:** O(nlog(n)), where n is the number of vertices in the graph which executes n times
- void printDijkstra(int source) | **Computational Complexity:** O(mlog(n)), where n is the number of vertices in the graph and m is number of edges in in the graph
- void printGraph() | **Computational Complexity:** O(n*mlog(m)), where n is MAX_VERTICES and m is the number of edges per vertex
- bool isVertex(int vertex) | **Computational Complexity:** O(1)