Benjamin Reichler and Aaron Shih

Problems encountered: Originally, Min-heap functions were declared and written at the bottom of Main.cpp, but were moved to the top of the file so that all functions were declared before being called. Also, heapify() did not originally check to make sure that 2*i is a valid index in the input array, leading to segmentation faults. Finally, while originally programming Dijkstra's algorithm, visited[] was never updated, so nodes no longer in minHeap[] were still marked as not visited; this was fixed by marking visited[currMin] = true after currMin was popped from minHeap[]. Also, since adjmatrix[][] was written to be 0-indexed, meaning information about vertex 1 existed in the row with index 0, but other arrays were written to be 1-indexed, there was some confusion during coding about which element to access in any given array.

Possible bugs: The program does not check to make sure that u and v are less than or equal to the number of nodes in G, n. It also does not check that n is greater than 0. Additionally, instead of checking that the input file has exactly m+1 lines, the program will simply take input integers until the end of the file is reached. For these reasons, the behavior of the program given a file that does not conform to the input format specifications in the project overview document (a malformed input document) may be unintuitive or unpredictable.

External references: Only slide decks presented in Professor Luo's CSE 310 class this semester were used as references to design the Min-Heap, Adjacency Matrix, and Dijkstra's Algorithm.

Design Decisions: Main.cpp builds the adjacency matrix by setting the 2D integer array adjmatrix[][] at both row u-1, column v-1, and row v-1, column u-1 equal to weight_uv for each input line of the form "u v weight_uv" after the first line, and then iterates through adjmatrix to display each weight. Main.cpp also constructs an array vertices[] of Vertex objects, initializes a vertex's index and sets its degree to 1 the first time that vertex appears in the input, and increments its degree on every subsequent appearance. After reading all edges, the program loops through the vertices to identify and collect all vertices with odd degrees. For each odd-degree vertex, Main.cpp runs Dijkstra's algorithm by first resetting every vertex's distance to INT_MAX, setting the source vertex's distance to 0, and using a custom min-heap of Vertex* objects to extract the unvisited vertex with the smallest distance repeatedly. The heap is rebuilt whenever distances change to ensure the minimum element remains at the root. This algorithm then relaxes all unvisited neighbors using the adjacency matrix. When each run of Dijkstra's completes, the program outputs the shortest-path distances from the current odd-degree vertex to all other vertices.