Cameron Matsui
Data Structures Final Project

About: My final project compares the runtimes of Prim's and Dijkstra's algorithms between matrix and adjacency-list implementations of weighted graphs. Dijkstra's algorithm finds the shortest path from some starting vertex (always 0 in my project) to all other vertices and its results are stored in a table; in the case that another vertex in the graph is unreachable from the starting vertex, this distance is marked as infinity. Prim's algorithm finds the minimum spanning tree in a connected, undirected graph. Note that runtimes should not be compared between algorithms, but between graph implementations. My Prim's implementation happens to take much more time than my Dijkstra's implementation, but I have verified that both are correct and reasonable implementations.

Data: My program generates a random graph for each algorithm with a number of vertices, n, that can be designated by the user (default is 7500) . For Dijkstra's algorithm, a directed, not necessarily connected, graph is filled by adding a random edge $4n$ times. For Prim's algorithm, an undirected, connected graph with approximate $3n$ edges is created. This graph is guaranteed to be connected because the first phase of generation effectively takes a random walk through each vertex. To make the minimum spanning tree more interesting than the graph itself, the second phase randomly adds $2n$ edges. The choices of the numbers of edges were arbitrary.

Data Structures: My program implements graphs, both backed by a matrix and adjacency-lists, as well as a priority queue backed by a binary heap and a list backed by a doubly-linked list. The graphs are used to run Dijkstra's and Prim's algorithms on, the priority queue is used in the algorithms to find the vertex with the minimum distance, and the lists are used in the adjacency-lists graph as well as to return lists of in-edges and out-edges for a given vertex for both graph implementations.

Instructions: There are two files with main methods, RunMatrix.java and RunLists.java, which test matrix and adjacency-lists implementations respectively. These programs print out runtimes to the console, and also log the graphs that they generate (represented in adjacency-lists form), the results of the algorithms, and the runtimes in the "Source/Logs" folder. Results can be interpreted by comparing runtimes between implementations from the logs. Note that since the graphs are randomly generated each time, these programs will not be running on the same graphs, but the graph sizes are large enough that it is reasonable to compare the two implementations from just a few executions of each.

Future: First, I would like to explore the difference in actual (ms) runtimes between the two algorithms in my implementations. Again, I have verified that the algorithms are correctly implemented but am still finding that Prim's algorithm runs markedly slower. Otherwise, if I were to extend my project in the future, I would like to include some sort of visualization, though this would be hard to do with random graphs. Also, though Dijkstra's and Prim's algorithms are graph algorithms, they raise interesting questions about the various implementations of priority

queues; I would explore this further by building a binomial heap and Fibonacci heap and comparing runtimes between these, the binary heap that I created, and a linked-list based priority queue.

Acknowledgements:

- Wikipedia pages on Dijkstra's and Prim's algorithms
- A couple of great YouTube videos on each algorithm:
    - youtube.com/watch?v=pVfj6mxhdMw&t=232s
    - youtube.com/watch?v=Uj47dxYPow8
- Open Data Structures in Java (specifically, the chapters on binary heaps and graphs)
- W3School's article on file creation and writing in Java
    - w3schools.com/java/java_files_create.asp
- The Java documentation at docs.oracle.com, particularly on generics and the comparable interface
- Tamir, Morrison, and Rinetzky: A Heap-Based Concurrent Priority Queue with Mutable Priorities for Faster Parallel Algorithms (to understand the problem of modifying priorities in a binary heap)
    - https://www.cs.tau.ac.il/~mad/publications/opodis2015-heap.pdf