Andy Drafahl, acd7, 311 HW4

1. The Java type you chose to represent the adjacency list inside your graph class.
   a. The graph is implemented using a Map<T, List<Pair<T, Long>>>. Each vertex (of generic type T) is mapped to a list of neighbors using Pair<T, Long>, where the Pair holds a neighbor vertex and the weight of the edge.

2. How you assigned edge weights when converting the image to a graph.
   a. Edge weights are based on pixel brightness. For two adjacent pixels, the average intensity is calculated using RGB values. The cost is computed as: return 5 + (255 - avgIntensity) / 8

3. How you solved the proposed problem using a constant number of invocations of Dijkstra's algorithm.
   a. I used dummy source nodes for both the rows and the columns that were connected to all leftmost and uppermost pixels, respectively. That way, Dijkstra could just start from that external source and it would work across the entire graph. This got my total calls down to 2.

4. The heap implementation you used. If you did not write it yourself and it is not part of the standard Java Class Library, you should mention this, including the license of the code and a link to it.
   a. I used the PriorityQueue<Pair<T, Long>> from the standard Java Class Library (java.util.PriorityQueue).