By Anando Zaman

Exercise 4.4.25 Shortest Path between two subsets: Given a digraph with positive edge weights, and two distinguished subsets of vertices S and T, find a shortest path from any vertex in S to any vertex in T. Your algorithm should run in time proportional to **E log V**, in the **worst case**.

Solution: Since the complexity is $ElogV$, we use Dijkstra's shortest path algorithm.

We can use a modification of Dijkstra's algorithm code to solve this. The original Dijkstra Algorithm code can be found on page 655 on the Princeton Algorithms 4th edition textbook.

Modified Algorithm description:

1. Step 1: Initialize the data structures for Dijkstra's algorithm(Minimum Priority Queue, array to keep track of total distances and array for previous edges connected to a node): DirectedEdge[] edgeTo; double[] distoTo; IndexMinPQ¡Double pq¿; int closestVertexInSubsetT

2. Step 2: With the edgeweighted digraph, we first set the $distTo[]$ for all vertices to $POSITIVE\_INFINITY$ initially so that comparisons can be done easily to relax the edges later.

3. Step 3: Next, we go through subset S and assign all of the vertices in our $distTo[]$ array to equal 0. This is because these vertices in set S will represent our set of "source" vertices. We will be trying to find single source shortest path from all of these vertices and pick the first one that makes a valid path.

4. Step 4: Insert the subset S vertices into the priority queue(aka IndexMinPQ) with a starting distance of 0.0 since no distance yet computed from these nodes to any other node.

5. Step 5: Next, while our priority queue is NOT empty, pop the minimum vertex in terms of least distance from the Priority Queue.

6. Step 6: Find all adjacent edges to min-vertex previously popped off. "Relax" these edges. The process of "relaxing" edges is checking if a cheaper paths can be formed from the current vertex to its adjacent nodes. Note; that the distances are still relative to the "source" node despite picking vertices adjacent to whatever the current node is on. If a cheaper path can be formed, then remove the old path and replace it with the new cheaper edges for the path.

7. Step 7: To relax edges, use $edge.from()$ and $edge.to()$ to get the starting and ending vertices of an edge. If $EndVertex\ distance > (StartVertex\ distance + edge.weight())$, then update path to reflect

the new path which consists of $edge.from()$ along with the edge.weight() value. In other words, if the old path was more expensive than the current path we found($edge.from()$ + edge.weight()), then update the path since a cheaper path is found.

8. Step 8: After relaxation, we can check if subsetT contains the minimum vertex that we got from step 5. If true, then we found closest vertex in subsetS that is also contained in subsetT. We can break out of the loop. Otherwise, continue dijkstras and relax every other edge taken from MinPQ until there is an min edge that is contained in both S and T. If nothing found, then no path exists between S and T.

9. Step 9: The edgeTo array variable declared earlier above is contain the parent nodes of each edge. We can use this to backtrack from the final edge all the way back to the original edge and vertex which will form out path. Similar to how backtrack to find path in DFS and BFS algorithm code.