W10 - Data Structure & Algorithms

Problem 1: Find the Shortest Path

Overview: Implement a program to find the shortest path from the first city (index 0) to the last city (index n - 1) given a list of direct distances between cities in a 2D array. If there is no direct path between two cities, the distance is represented as zero.

Algorithm Steps:

1. **Graph Representation**:
   - Represent the cities and distances using a graph where the nodes are the cities, and the edges are the distances between the cities.
2. **Dijkstra's Algorithm**:
   - Use Dijkstra's algorithm to find the shortest path from the first city to the last city. This algorithm is efficient for graphs with non-negative edge weights.
3. **Priority Queue for Efficient Searching**:
   - Use a priority queue to efficiently select the next city with the shortest distance from the start city.
4. **Handling Edge Cases**:
   - Ensure that the graph is properly constructed and that distances are non-negative.
5. **Code**

```java
import java.util.Arrays;
import java.util.PriorityQueue;

public class ShortestPathCalculator {
   // Function to find the shortest path from the first city to the last city
   public static int shortestPath(int[][] distances) {
       int n = distances.length;
       int[] dist = new int[n];
       Arrays.fill(dist, Integer.MAX_VALUE);
       dist[0] = 0;

       PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> dist[a] - dist[b]);
       pq.offer(0);

       while (!pq.isEmpty()) {
           int city = pq.poll();

           for (int i = 0; i < n; i++) {
               if (distances[city][i] > 0 && dist[i] > dist[city] + distances[city][i]) {
                   dist[i] = dist[city] + distances[city][i];
                   pq.offer(i);
               }
           }
       }
       return dist[n - 1];
   }

   // Main method to test the shortestPath function
   public static void main(String[] args) {
       int[][] distances = {
           {0, 3, 2, 0},
           {3, 0, 0, 5},
           {2, 0, 0, 9},
           {0, 5, 9, 0}
       };
       System.out.println("The shortest path's length from city 0 to city " + (distances.length - 1) + " is: " -
    }
}
```

Problem 2, 3

↳ *3 cells hidden*