Emily Monroe and Ethan Smith

CS 355

Fall 2024

Project 3

Due Date: November 15, 2024

Program Requirements:

Create a real-world project that demonstrates the shortest path problem. Your project will be composed of a front-end application that makes use of a back end. Your back end will solve the shortest path algorithm on a directed acyclic graph. The front end will show you that you back-end works. You will have to create a proposal, a design document, implementation, presentation, a final project portfolio and a team assessment.

Program Inputs:

Emergency Locations

- Wesleyan hall coordinates
- The Commons coordinates
- Flowers Hall Coordinates

Hospital Locations

- North Alabama medical center Coordinates
- Hellen Keller Hospital Coordinates

Paths

- At least 2 different routes to a hospital per build ng

Test Plan:

Test Case 1.1: Add Edge

Description: Verify that Graph::addEdge correctly adds an edge to the adjacency list.

- **Steps**:
  1. Create a graph instance.
  2. Add an edge between nodes 1 and 2 with a weight of 5.
  3. Retrieve neighbors of node 1.
- **Expected Result**: The neighbor list of node 1 should contain node 2 with weight 5.

Test Case 1.2: Get Neighbors

Test Case 3.2: Find Shortest Path to Nearest Hospital

Description: Ensure the shortest path from an emergency location to the nearest hospital is found.

- **Steps:**
    1. Create a graph with several nodes and edges.
    2. Designate node 4 as a hospital and node 1 as an emergency location.
    3. Run EmergencyService::getShortestPath for the emergency location.
- **Expected Result:** The path should correctly list the nodes and total distance to the nearest hospital.

Test Case 4.1: Multiple Hospitals and Emergency Locations

Description: Test with multiple hospitals and emergency locations in a complex graph.

- **Steps:**
    1. Create a graph with nodes and multiple edges.
    2. Add several hospitals and emergency locations.
    3. Run the program to determine the shortest path to the closest hospital for each emergency location.
- **Expected Result:** Each emergency location should output the correct shortest path and distance to the nearest hospital.

Test Case 4.2: Disconnected Graph Segments

Description: Test if the program handles cases where an emergency location has no path to any hospital.

- **Steps:**
    1. Create a graph with two disconnected segments: one containing hospitals and the other containing emergency locations.
    2. Try to find the shortest path for an emergency location in the segment without hospitals.
- **Expected Result:** The program should return an error or indicate that no hospital is reachable.

Test Case 4.3: Edge Cases with One Node

Description: Test if the program behaves correctly with a single node graph.

- **Steps:**
    1. Create a graph with only one node designated as both a hospital and an emergency location.

2. Run the program to find the shortest path.

- **Expected Result**: The output should indicate zero distance or a message that the hospital is at the emergency location.

Algorithm In Pseudocode:

Define Classes

Class: Graph

- **Attributes**:
  - numNodes: Integer, total number of nodes in the graph.
  - adjList: Array where each node has a map representing neighboring nodes and edge weights.
- **Methods**:
  - **Method**: addEdge(u, v, weight)
    - Adds a directed edge from node u to node v with the specified weight.
    - **Input**: u (node), v (node), weight (integer)
    - **Process**:
      1. If u is not in adjList, add it as a key with an empty list.
      2. Append (v, weight) to the adjacency list of u.
  - **Method**: getNeighbors(node)
    - Returns the list of neighbors and weights for a given node.
    - **Input**: node (integer)
    - **Output**: Map (neighbor, weight)

Class: Dijkstra

- **Method**: findShortestPath(graph, start)
  - Uses Dijkstra's algorithm to calculate the shortest path from the start node to all other nodes in the graph.
  - **Input**: graph (Graph instance), start (integer)
  - **Output**: Map of shortest distances from start to each node.
  - **Algorithm**:

1. Initialize a distance map with all nodes set to infinity, except start, which is set to 0.
2. Initialize a min-heap (priority queue) with (0, start).
3. While the priority queue is not empty:
    - Extract the node with the smallest distance currDist and currNode.
    - If currDist is greater than distance[currNode], continue (outdated entry).
    - For each neighbor, weight in Graph::getNeighbors(current_node):
        - Calculate newDist = currDist + weight.
        - If newDist is less than distance[neighbor]:
            - Update distance[neighbor] to newDist
            - Insert (newDist, neighbor) into the priority queue.
4. Return distance dictionary.

- **Method**: findClosestHospital(distances, hospitalNodes)
    - Finds the nearest hospital from the given distances dictionary.
    - **Input**: distances (Map), hospitalNodes (set of Nodes)
    - **Output**: Nearest hospital node (integer)
    - **Algorithm**:
        1. Initialize minDist to infinity and closestHospital to null.
        2. For each hospital in hospitalNodes
            - If distances[hospital] is less than minDist
                - Update minDist to distances[hospital].
                - Set closestHospital to hospital.
        3. Return closestHospital

Class: EmergencyService

- **Attributes**:
    - graph: Instance of Graph.

- hospitalNodes: Node array representing hospital nodes.
- emergencyNodes: Node array representing emergency location nodes.

- **Methods**:
  - **Method**: addHospital(node)
    - Adds a hospital at the specified node.
    - **Input**: node (Node)
    - **Process**: Add node to hospitalNodes.
  - **Method**: addEmergeLoc(node)
    - Adds an emergency location at the specified node.
    - **Input**: node (Node)
    - **Process**: Add node to emergencyNodes.
  - **Method**: getShortestPath(emergencyNode)
    - Finds the shortest path from an emergencyNode to the nearest hospital.
    - **Input**: emergencyNode (Node)
    - **Output**: List of nodes in the shortest path to the nearest hospital.
    - **Algorithm**:
      1. Use Dijkstra::findShortestPath(graph, emergencyNode) to get distances from emergencyNode.
      2. Use Dijkstra::findClosestHospital(distances, hospitalNodes) to find the closest hospital node.
      3. If closestNospital is null, return "No hospital reachable".
      4. Otherwise, reconstruct the path from emergencyNode to closestHospital using the distances and return it.

Main Program

1. Initialize a Graph instance and an EmergencyService instance with the graph.
2. Load or add edges to the graph.
3. Add hospitals and emergency locations as needed.
4. For each emergency location:

- Call getShortestPath(emergency_node) to get the path and print it.

Algorithm in Flowchart: