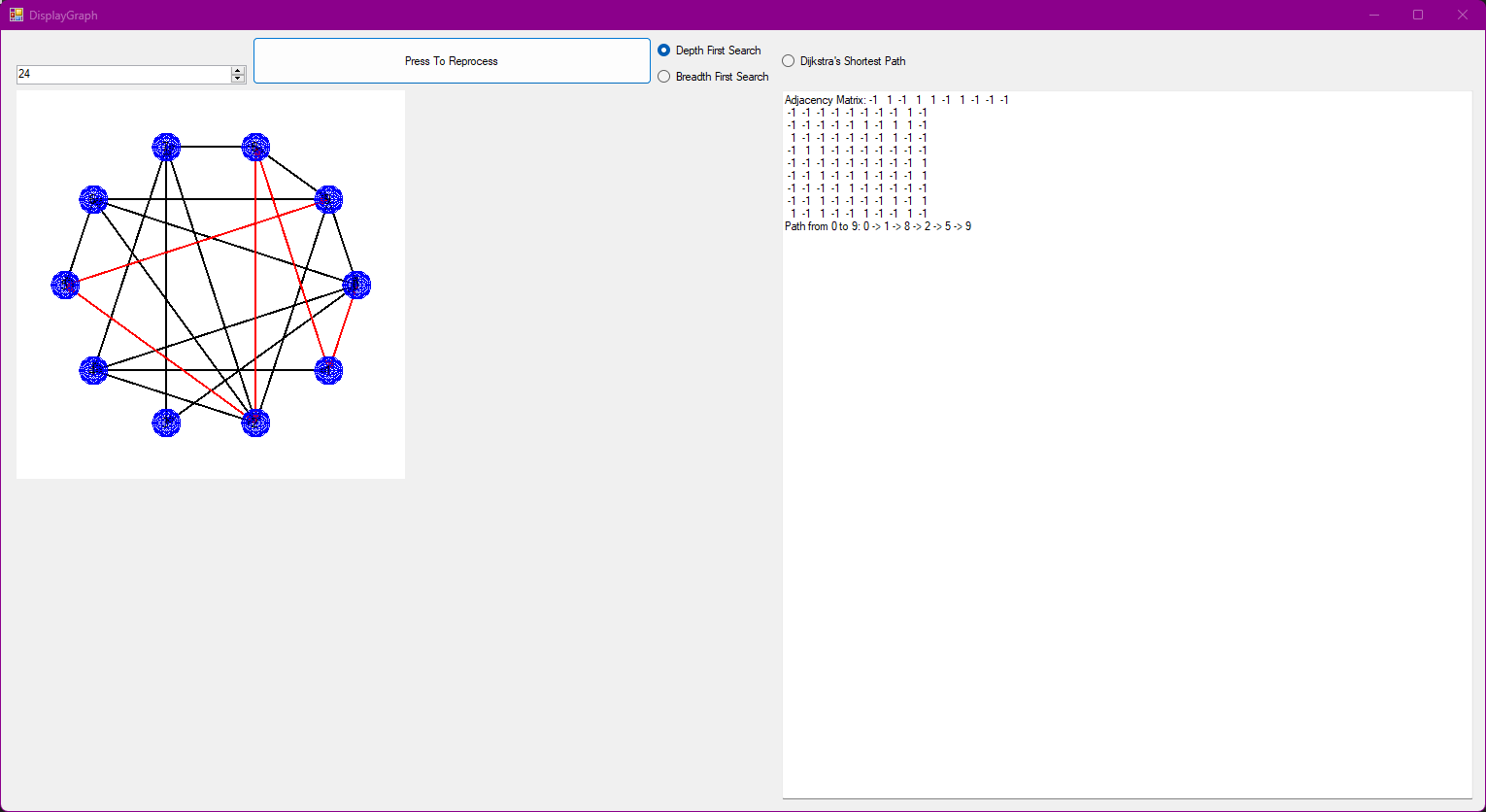
Isabella Dougherty

1. **Depth-First Search (DFS) What is the length of the path your depth-first search encountered on its way to locating vertex 9? How many nodes were visited in reaching vertex 9:**

* Path Length: The path from vertex 0 to vertex 9 consists of 4 edges: (0 to 2, 2 to 5, 5 to 8, 8 to 9).
* Total Nodes Visited: 9 nodes (vertices 0 through 9, excluding any unreachable nodes).
* Nodes Visited to Reach Vertex 9: All nodes visited before reaching vertex 9, which may be fewer if vertex 9 is found earlier in the traversal.
* **Time Complexity of Depth-First Search (DFS):**
* Best Case Complexity: O(n)
* Worst Case Complexity: O(n + m)
* Average Case Complexity: O(n + m)
  + Where **n** is the number of nodes (vertices) and **m** is the number of edges in the graph.
* **Justification:**
* **Vertices (V) and Edges (E)**: V represents the number of vertices, and E represents the number of edges in the graph.
* **Traversal**: DFS visits every vertex and explores every edge in the connected component containing the starting vertex.
* **Worst Case Analysis**:
  + In the worst case, the graph is dense (many edges), and DFS must explore all nodes and all edges.
  + Time Complexity: O(n + m)
* **Best Case Analysis**:
  + In the best case, the graph is sparse (few edges), but DFS still needs to visit all connected nodes.
  + Time Complexity: O(n)
* **Average Case Analysis**:
  + On average, DFS visits a subset of nodes and edges depending on the graph's connectivity.
  + However, the time complexity is still bounded by O(n + m).
* **Impact of Graph Density**:
  + Sparse Graphs: Fewer edges (“small m”), so the time complexity is closer to O(n).
  + Dense Graphs: More edges (“large m”), increasing the time spent exploring edges. Time complexity approaches O(n + n²) because, in a fully connected graph, m can be up to O(n²).
* **Simplifying the Complexity**:
  + In a dense graph where m is proportional to n² (i.e., m = O(n²)), the time complexity becomes O(n²).
  + However, for most practical purposes, the time complexity is expressed as O(n + m).
* **Conclusion**:
  + The Big O notation expresses the upper bound on the time complexity.
  + Since DFS must, in the worst case, explore all vertices and edges reachable from the starting vertex, the time complexity is O(V + E) for all cases.
  1. 

1. **What is the BigO (best, worse, average) of your depth-first search? Carefully and fully justify your analysis.**
   1. Time Complexity of Depth-First Search (DFS):
      1. Best Case Complexity: O(n)
      2. Worst Case Complexity: O(n + m)
      3. Average Case Complexity: O(n + m)
         * n is the number of nodes (vertices) in the graph.
         * m is the number of edges in the graph.
   2. Justification:
      1. V represents the number of vertices, and E represents the number of edges in the graph.
      2. DFS visits every vertex and explores every edge in the connected component containing the starting vertex.
         * Worst Case Analysis:
           1. In the worst case, the graph is dense (many edges), and DFS must explore all nodes and all edges.
           2. Time Complexity: O(n + m)
         * Best Case Analysis:
           1. In the best case, the graph is sparse (few edges), but DFS still needs to visit all connected nodes.
           2. Time Complexity: O(n)
         * Average Case Analysis:
           1. On average, DFS visits a subset of nodes and edges depending on the graph's connectivity.
           2. However, the time complexity is still bounded by O(n + m).
         * Impact of Graph Density:
           1. Sparse Graphs:

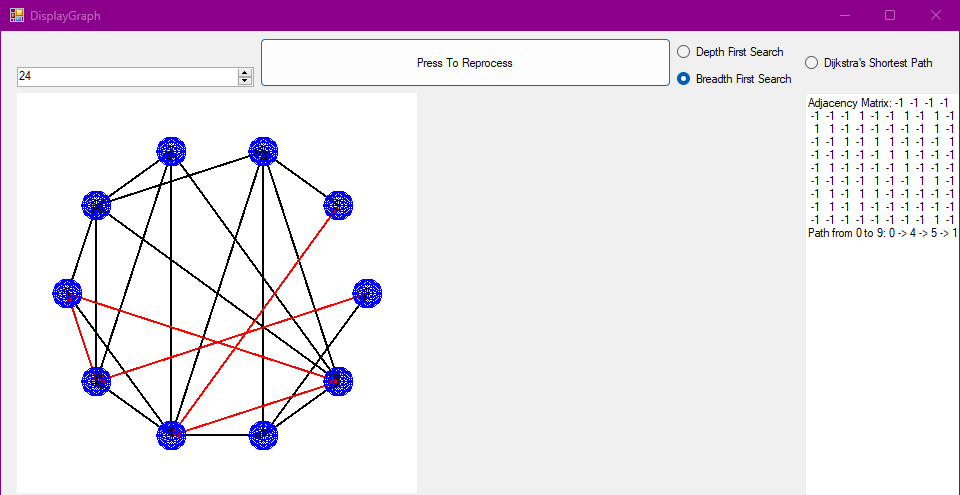
Fewer edges (small m), so the time complexity is closer to O(n).

* + - * 1. Dense Graphs:

More edges (large m), increasing the time spent exploring edges.

Time complexity approaches O(n + n²) because, in a fully connected graph, m can be up to O(n²).

* + - * Simplifying the Complexity:
        1. In a dense graph where m is proportional to n² (i.e., m = O(n²)), the time complexity becomes O(n²).
        2. However, for most practical purposes, the time complexity is expressed as O(n + m).
  1. Conclusion:
     1. The Big O notation expresses the upper bound on the time complexity.
     2. Since DFS must, in the worst case, explore all vertices and edges reachable from the starting vertex, the time complexity is O(n + m) for all cases.

1. 
2. **Assuming each edge is length 1, what is the shortest distance between node 0 and node 9? (The answer may be infinity.)**
   1. In the context of BFS on an unweighted graph where each edge has a length of 1:
      1. Shortest Distance: The shortest distance between vertex 0 and vertex 9 is the minimum number of edges connecting them.
      2. Shortest Distance: 2 (edges between 0 and 4, and 4 and 9)
3. **What is the Big O (best, worst, average) of your breadth-first search? Carefully and fully justify your analysis.**
   1. Time Complexity of Breadth-First Search (BFS):
      1. Best Case Complexity: O(n)
      2. Worst Case Complexity: O(n + m)
      3. Average Case Complexity: O(n + m)
         * Justification:
           1. BFS Algorithm Steps:

Initialization:

Mark all nodes as unvisited.

Initialize a queue.

Time Complexity: O(n)

Traversal:

Dequeue a node, visit it, and enqueue all its unvisited adjacent nodes.

Each node is enqueued and dequeued exactly once.

Time Complexity:

Visiting all nodes: O(n)

Exploring all edges: O(m)

* + - * 1. Best Case Analysis:

In the best case, the graph is sparse, and BFS finds the target node early.

Time Complexity: O(n)

* + - * 1. Worst Case Analysis:

In the worst case, BFS must explore all nodes and all edges to find the target node or confirm it's unreachable.

Time Complexity: O(n + m)

* + - * 1. Average Case Analysis:

Typically, BFS explores nodes level by level, which may result in visiting a significant portion of the graph.

Time Complexity: O(n + m)

* + - * Impact of Graph Density:
        1. Sparse Graphs:

Fewer edges, so fewer adjacent nodes to enqueue.

Time complexity is closer to O(n).

* + - * 1. Dense Graphs:

More edges, resulting in more adjacent nodes to enqueue and process.

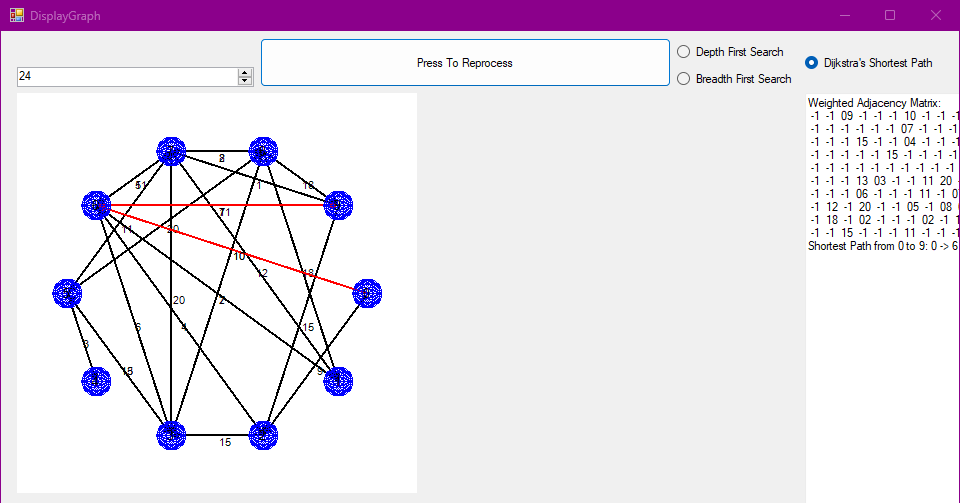
Time complexity approaches O(n + n²), as m can be up to O(n²) in a fully connected graph.

* + - * Conclusion:
        1. Breadth-First Search has a time complexity of O(n + m), where:

n is the number of nodes.

m is the number of edges.

This indicates that the algorithm's running time depends linearly on the number of nodes and edges in the graph.

1. 
2. **What is the Big O (best, worst, average) of your shortest path algorithm search? Carefully and fully justify your analysis. How does the percent of your initial matrix fill affect your Big O analysis?**
   1. Time Complexity of Dijkstra's Algorithm (without a Priority Queue):
      1. Best Case Complexity: O(n²)
      2. Worst Case Complexity: O(n²)
      3. Average Case Complexity: O(n²)
         * Justification:
           1. Algorithm Steps:

Initialization:

Set distances to all nodes as infinite except the source node.

Time Complexity: O(n)

Main Loop:

For each node:

Find the node with the minimum tentative distance that hasn't been processed yet.

Time Complexity: O(n) per iteration (since we may need to scan all nodes).

Relaxation Step:

Update distances of all adjacent nodes if a shorter path is found.

Time Complexity: O(n) per iteration (since we may need to check all nodes).

Total Time Complexity:

The main loop runs n times.

Each iteration takes O(n) time for the minimum distance search and O(n) for the relaxation step.

Total Time Complexity: O(n²)

* + - * Impact of Graph Density:
        1. Sparse Graphs:

Fewer edges (m is small), so the relaxation step may be faster in practice.

However, since we check all nodes during the minimum distance search and potentially during relaxation, the time complexity remains O(n²).

* + - * 1. Dense Graphs:

More edges (m approaches n²), increasing the time spent in the relaxation step.

The time complexity is still O(n²), but with a larger constant factor due to more edges.

* + 1. Conclusion:
       - Dijkstra's Algorithm without a priority queue has a time complexity of O(n²), where:
         1. n is the number of nodes.
         2. This means the algorithm's running time grows quadratically with the number of nodes in the graph.
         3. Effect of Matrix Fill Percentage on Big O Analysis:
         4. Higher Fill Percentage (Denser Graphs):
         5. Increases the number of edges (m), potentially leading to more work during the relaxation step.
       - However, the dominant factor in the time complexity is the O(n²) from the nested loops over nodes.
         1. Lower Fill Percentage (Sparser Graphs):

Decreases the number of edges (m), which may reduce the actual running time.

The Big O notation remains O(n²), but the algorithm may run faster in practice due to fewer edges to process.