Summary Part

1. **In your own words, compare and contrast BFS with Dijkstra’s algorithm.**

   Breadth-First Search (BFS) and Dijkstra’s algorithm are both graph traversal methods but are suited for different types of problems. BFS is ideal for unweighted graphs or graphs where all edges have the same weight. It explores all nodes level by level, starting from a given node, ensuring that all nodes at the current depth are visited before moving to the next level. This makes BFS effective in finding the shortest path in unweighted graphs by the nature of its traversal. In contrast, Dijkstra’s algorithm is designed to handle weighted graphs with non-negative weights. It finds the shortest path from a starting node to all other nodes by using a priority queue to always extend the shortest known path to a node. This makes Dijkstra’s algorithm more versatile for real-world problems involving varying edge weights, such as road networks or network routing.

2. **What is the purpose of Dijkstra's algorithm, and in what types of problems is it commonly used?**

   Dijkstra's algorithm is designed to determine the shortest path from a starting node to all other nodes in a graph with non-negative edge weights. It is used in optimizing travel routes in transportation systems.

3. **What does it mean for a node to be reachable from another node in a graph?**

   A node B is said to be reachable from node A if there is a path consisting of one or more edges that connect A to B. In other words, you can traverse from A to B by following the edges of the graph.

4. **Why is tracking the "predecessor" node in Dijkstra's algorithm necessary?**

   By keeping track of the predecessor nodes, we can reconstruct the shortest path from the starting node to any other node after the algorithm has completed. This is essential for applications that require knowledge of the specific path taken, not just the distance.

5. **What role does the 'infinity' concept play in Dijkstra's algorithm?**

   Initially, the distances to all nodes from the starting node are set to infinity to signify that they are not yet reachable. As the algorithm progresses, these infinite values are updated with actual distances as shorter paths are discovered. The use of infinity ensures that any found path will be shorter than an infinite distance, thus correctly identifying reachable nodes and updating their shortest paths.

Self-reflection part

What did you learn?

This week, I learned the Dijkstra’s algorithm.

What went smoothly?
Understanding the theoretical foundations of Dijkstra’s algorithm. Implementing these algorithms in code.

What was difficult about the content this week?
The challenging part was comprehending the nuances of Dijkstra's algorithm, particularly how the priority queue is managed and updated during the process.

How will you approach things differently next time?
I plan to spend more time on visual aids and simulations to better understand the dynamic changes occurring during the execution of these algorithms.

Do you have any feedback about the content for this week?
Good!