CS-1203 - Spring 2023 - Assignment 4

Anirvaan Kar

Question 1

This code is suitable for directed graphs as well. as we would just have to remove one half (as it is a symmetric matrix for a directed graph sided we could just have the other direction be set to undefined)

The time complexity of Dijkstra's algorithm is $O(V^2)$ for a dense graph and $O(E \log V)$ for a sparse graph, where V is the number of vertices and E is the number of edges in the graph. The auxiliary space required is $O(V)$ to store the distance values of vertices.

To enhance the time complexity, a min-priority queue can be used instead of a linear search to find the vertex with the smallest tentative distance. The algorithm based on this idea involves initializing distance values and a priority queue, inserting the source node into the priority queue with distance 0, and iteratively extracting the node with the minimum distance from the priority queue, updating the distances of its neighbors, and inserting updated neighbors into the priority queue.

Therefore, the Time Complexity becomes $O((V + E) \log V)$. This includes the initialization ($O(V)$) and the main loop ($O(V \log V + E \log V)$). The main loop runs V times, and in each iteration, the minimum distance vertex is extracted from the priority queue, taking $O(\log V)$ time. The relaxation operations for adjacent vertices contribute $E \log V$ operations.

The time complexity and auxiliary space complexity mentioned ($O((V + E) \log V)$ and $O(V)$, respectively) correspond to an optimized version of Dijkstra's algorithm using a priority queue or a min-heap data structure.

The relevant data structures for implementing Dijkstra's algorithm include the adjacency matrix (a 2D array representing edges and weights), adjacency list (an array of linked lists representing vertices and their neighbors), and a min-priority queue for efficient key-based operations.