Assignment 4
COSC 2P03
Ameya Chindarkar
#7023609

This program made use of graphs, which was implemented in the class “Graph”. This program reads a maze from the user or an input file containing a maze, and constructs a graph representation of the maze, and provides methods to calculate the shortest path and its cost. This class implements the Comparable interface. This allows instances of this class to compare the weight of 2 edges.
Next, a constructor “public Graph (char[][][] input) which takes the input of a 3d array called ‘input’. In this constructor, The input lengths are being stored for each dimension and by multiplying these, I find the number of Nodes. I initialize the adjacency list “adjList’ as an array list, where each index represents a node.
The matrix is iterated over to create the graphFor each cell in the matrix, it determines the node number based on its layer, row and column. If a wall is encountered the iteration continues to the next cell. It checks neighboring nodes in all directions (north, south east, west, up and down) and adds connections between the current node and neighboring nodes if they are valid cells (not with X).
Under this class is also the ShortedPath() method, which implements Dijkstra's algorithm to find the shortest path in the graph, from the start ‘s’ to end ‘e’. Then I create a distance and parents array, to store cumulative pairs. Then I create a min heap which initializes the start node “s” distance as 0 and adds it to the queue. It then iteratively extracts the node with the min distance from the queue. If a shorter path is found to an adjacent node, it updates the distance, parent and adds the node to the queue. Returns true if the destination is reachable.
Next method is to get the cost of the shortest path, getShortestPathCost(): which checks if the shortest path has been calculated. If not, it calls the shortedPath method. It returns the shortest path.
The getShortestPath() method is similar to getShortestPathCost(). It retrieves the shortest path by traversing the parents array starting from the destination node (d). It adds the corresponding directional characters (n for north, u for up…) to a list. Finally it reverses the list to obtain the correct order of directions from the start to end.