TASK 3

Maze Solver Using Dijkstra's Algorithm (C++)

This project implements a maze solver in C++ using Dijkstra’s Algorithm to find the minimum cost path from a start point (S) to a goal (G) in a 2D weighted maze.

Features

Accepts maze input from standard input (cin)

Supports different terrain types with variable movement costs

Finds the shortest path using Dijkstra's algorithm

Handles obstacles (#) and weighted paths (., ~, ^)

Reports total cost to reach the goal or notifies if unreachable

Maze Format (Input)

Each line is a row of the maze. Maze cells can be:

Symbol	Meaning	Cost
S	Start position	0

G	Goal position	0
.	Normal path	1
~	Water (harder)	3
^	Mountain (tough)	5
#	Wall (blocked)	∞ (inaccessible)

Input ends with an empty line.

How the algorithm works

The given input of characters is stored as integers representing the cost of reaching there in a 2D matrix, and another matrix of the same dimensions stores whether the given node is visited or not. The wall's cost is taken as INT_MAX

The walls are considered visited as we cannot go to the node we have already visited.

If a node is visited, it means the shortest path of reaching there is known now, and there is no other smaller path we can use to reach there.

We may arrive at a node from various directions, but the smallest distance taken to reach it is the only one considered.

The total distance calculated for this purpose from going from node A to node B is the distance cost for reaching A + distance cost for reaching B.
This way, we calculate the shortest distance we can take to reach all the nodes given.
This code runs repeatedly until we reach the ending node, and the total distance count is calculated there.
The priority queue is updated in each iteration to tell which next node should be calculated.
The priority queue is a mean heap that tells the node with the minimum distance from the node we have, i.e., the node we must seek next.
Future changes
We can add a feature always to keep track of which node we were right before coming to this node, so when we get the smallest path of reaching there, we also get the node we followed to get there, and when we reach the destination, we will have all the nodes we traversed to get there in the shortest path.