**Assignment 4 – Implement the A Pathfinding Algorithm**

**Theory – A\* Search Algorithm**

The A\* (pronounced "A-star") algorithm is a popular and efficient pathfinding algorithm, widely used in fields like robotics, video games, and logistics. It's known for finding the shortest path between a starting point and a goal point on a graph or grid, avoiding obstacles along the way.

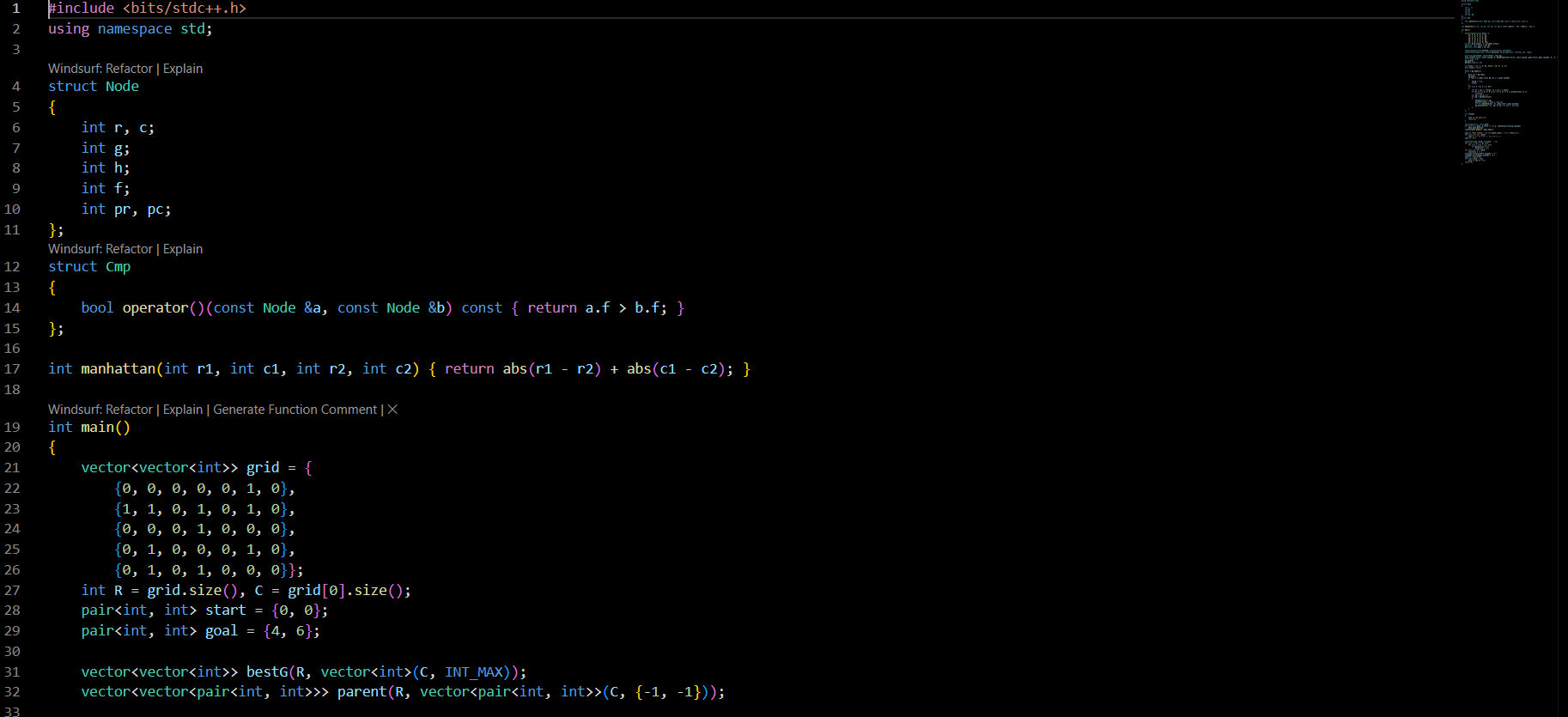
A\* is a "best-first search" algorithm, meaning it intelligently decides which path to explore next. It does this by calculating a "cost" for each potential step, using the formula:

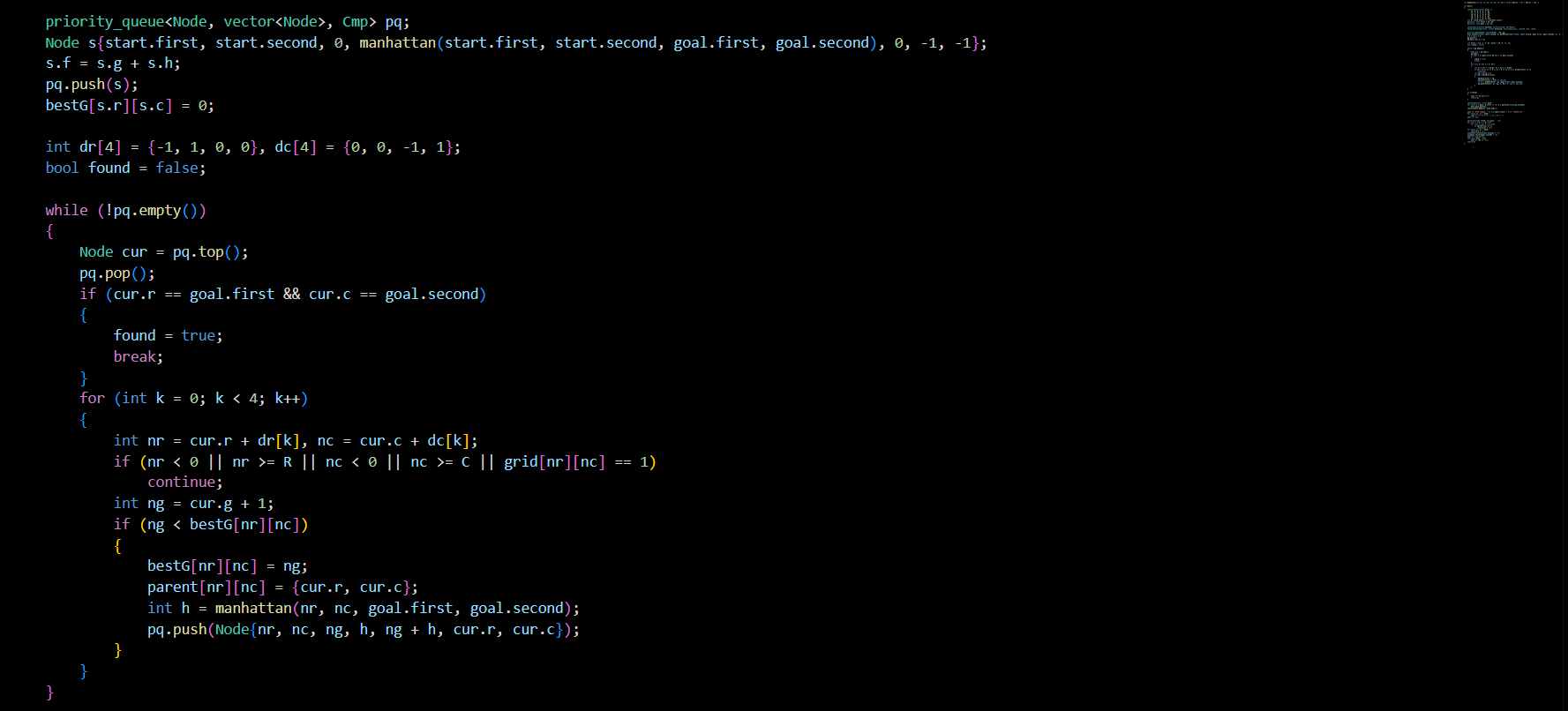
Where:

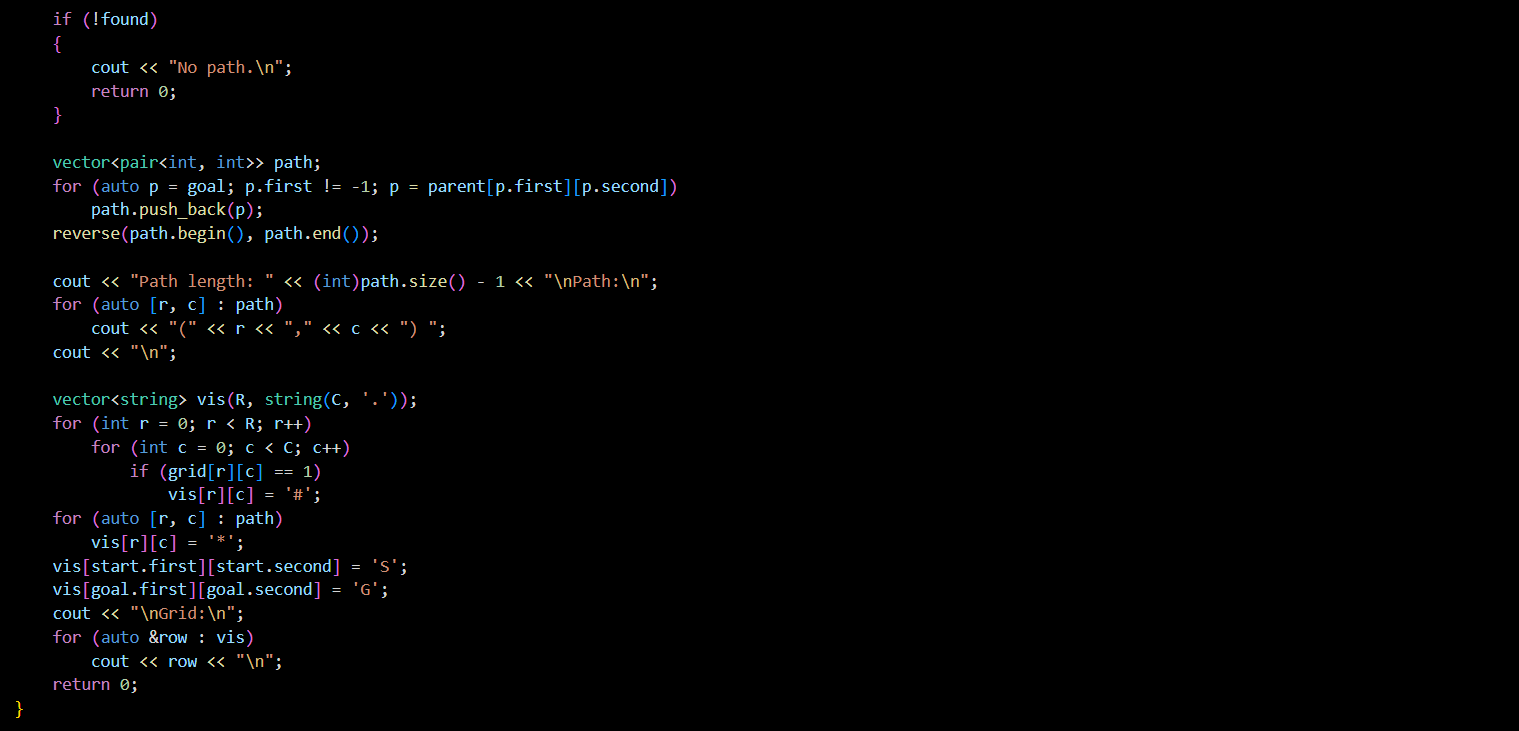
* is the next node on the path.
* is the actual cost of the path from the starting node to node .
* is the heuristic, which is an *estimated* cost from node to the goal. For the algorithm to find the shortest path, the heuristic must be *admissible*, meaning it never overestimates the actual cost. A common heuristic for grids is the Manhattan distance.
* is the total estimated cost of the path. A\* prioritizes exploring nodes with the lowest value.

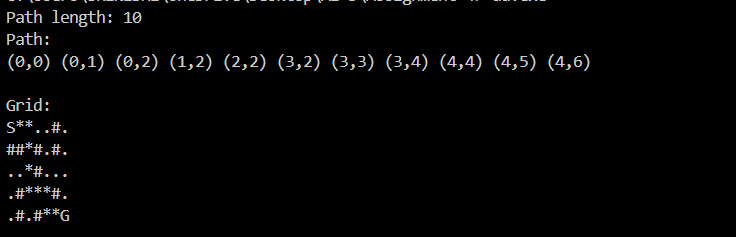
The algorithm maintains an "open list" (typically a priority queue) of nodes to visit, sorted by their score. It repeatedly selects the most promising node from this list, explores its neighbors, and updates their costs if a better path is found.

**Code –**

****

****

****

**Output - **

**Explanation –**

* Representation: The environment is a 2D grid where 0 is a walkable tile and 1 is an obstacle. The Node struct stores all necessary information for A\*, including its position (r, c), the g, h, and f costs, and its parent's coordinates for path reconstruction. A priority queue is used as the open list to efficiently retrieve the node with the lowest f score.
* Heuristic Function: The manhattan function calculates the distance between two points on a grid by summing the absolute differences of their coordinates. This is an admissible heuristic because it represents the shortest possible path if there were no obstacles.
* A\* Algorithm Logic:
  + Initialization: The algorithm starts by creating a bestG grid to store the lowest g score found so far for each cell. This prevents redundant exploration. The start node is created, its costs are calculated, and it is pushed into the priority queue.
  + Main Loop: The loop continues as long as the priority queue is not empty. In each iteration, it extracts the Node with the smallest f score.
  + Neighbor Exploration: It checks the four adjacent neighbors (up, down, left, right). It ignores any neighbor that is out of bounds or is an obstacle (grid[nr][nc] == 1).
  + Path Update: For a valid neighbor, it calculates a new g score (ng). The core of A\* is the condition if (ng < bestG[nr][nc]). If this new path to the neighbor is shorter than any previously found path, it updates bestG, records the parent, and pushes the new, improved Node into the priority queue.
* Path Reconstruction: Once the goal node is reached, the loop terminates. The parent grid is used to trace the path backward from the goal to the start. The resulting path is then reversed to get the correct order and printed.

**Conclusion -**

This program successfully implements the A\* pathfinding algorithm. It correctly navigates a 2D grid with obstacles to find the optimal path from a start to a goal location, showcasing a fundamental and powerful technique in artificial intelligence and computational geometry.