**Name: Kamran**

**Roll No: 002**

**Lab 7**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Program 1: A\* Search Algorithm (Shortest Path in Graph)**

**What is this Program?**

This program finds the shortest path between two nodes in a graph using the A\* search algorithm. It is a combination of Dijkstra’s algorithm and a heuristic function to guide the search efficiently.

**How does it work?**

Step 1: Initialize open and closed sets

- Open set stores nodes to be evaluated.

- Closed set stores nodes already evaluated.

Step 2: Set start node costs

- `g\_cost` keeps track of the cost from the start node to the current node.

- `parent` keeps track of the path.

Step 3: Choose current node

- From open nodes, choose the node with the lowest `g\_cost + h(n)` value.

- `h(n)` is the heuristic estimate of distance to the goal.

Step 4: Check goal

- If the current node is the goal, reconstruct the path from `parent` pointers.

Step 5: Update neighbors

- For each neighbor, update `g\_cost` and `parent` if a better path is found.

- Move evaluated nodes from open set to closed set.

Step 6: Repeat

- Continue until the goal is reached or no path exists.

Output Example

**Example Run:**

Shortest Path Found: ['A', 'B', 'D']

**Why this approach?**

1. Efficient Pathfinding: Uses heuristic to reduce the number of nodes evaluated.

2. Real Use: Used in AI, robotics, and games for pathfinding.

3. Beginner Friendly: Shows the combination of greedy search and shortest path logic.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_