

**Name: Aliza**

**Roll: Su92BSAIM051**

**Subject: AI LAB**

**Task :7**

**A\* Search Algorithm Documentation**

**Introduction**

This program uses the A\* (A-Star) search algorithm to find the shortest path between two nodes in a graph. It combines the real cost of moving from node to node with a guessed cost to reach the goal. A priority queue helps the program pick the best node to explore first, which makes the search faster and more efficient.

**Graph Structure**

The graph is saved as an adjacency list. Each node has a list of neighbors and the cost to move to them. The heuristic dictionary stores estimated distances from each node to the goal, helping the algorithm choose better paths.

**A\* Function Explanation**

The a\_star() method starts with the open list, cost dictionary g, parent dictionary, and a closed set. It removes the node with the lowest total cost (real cost + heuristic) from the priority queue. Then, it checks all neighboring nodes and updates their cost if a shorter path is found. When the end node is reached, the loop stops, and the path reconstruction function is used.

**Build Path Function**

The build\_path() method builds the final shortest path by following parent links from the destination back to the starting node. The order is then reversed so the path shows the correct travel direction.

**Key Features and Improvements**

The use of heapq makes choosing the lowest-cost node fast. Cost checks prevent updating worse paths. The closed set keeps the program from visiting the same node again, making the search faster and more accurate.

**Benefits**

This A\* method finds the shortest path quickly and uses fewer steps. It works well for maps, GPS systems, robot movement, and game characters because it avoids exploring unnecessary nodes.

**Conclusion**

Overall, this A\* implementation is simple, fast, and reliable. By using cost and heuristic values together, it can quickly find the best route in many real-world pathfinding situations.