Assignment No: - 4

Name :- Isha Ghorpade

Enrollnment :- 22420020

Roll :- 381066

Problem Statement

Implement the A* (A-star) algorithm for an application, such as shortest pathfinding in graphs, puzzle solving, or robot navigation.

Objectives

- To understand the working principle of the A* search algorithm.
- To apply heuristics in informed search.
- To implement A* for solving real-world problems (pathfinding, puzzles).
- To evaluate the algorithm in terms of optimality and efficiency.

Theory

Introduction

The A* (A-star) algorithm is an informed search algorithm that combines the advantages of Uniform Cost Search and Greedy Best-First Search. It is widely used for shortest pathfinding and graph traversal problems.

A* uses both:

- Cost so far (g(n)) → the actual cost from the start node to the current node.
- Heuristic (h(n)) → the estimated cost from the current node to the goal.

The evaluation function is:

f(n)=g(n)+h(n)f(n) = g(n) + h(n)

where

- f(n)f(n) → total estimated cost of the solution path through node *n*.
- g(n)g(n) → path cost so far.
- h(n)h(n) → heuristic estimate of cost from *n* to goal.

Working of A* Algorithm

1. Initialize an open list (nodes to be evaluated) and a closed list (already evaluated nodes).
2. Place the start node in the open list.
3. While open list is not empty:
    - Pick the node with the lowest f(n)f(n).
    - If it is the goal node → return path (solution found).
    - Else, expand it (generate neighbors).
    - Compute f(n)=g(n)+h(n)f(n) = g(n) + h(n) for each neighbor.
    - If neighbor not visited or found with lower cost → add/update in open list.
4. Continue until goal is reached or no path exists.

Heuristics in A*

- Admissible Heuristic: Never overestimates the cost (ensures optimality).
- Examples:
    - Manhattan Distance → for grid-based movement.
    - Euclidean Distance → for shortest path in continuous space.
    - Misplaced Tiles → for 8-puzzle problem.

Properties of A*

- Complete: Always finds a solution if one exists.
- Optimal: Finds the shortest path if the heuristic is admissible.
- Time Complexity: Depends on accuracy of heuristic (better heuristic → faster).
- Space Complexity: High, as it stores many nodes in memory.

Methodology

1. Define the problem as a graph or grid.
2. Represent nodes, edges, and costs.
3. Choose a heuristic function suitable for the application.
4. Implement A* algorithm using priority queue (min-heap).
5. Trace the path from start to goal.

Advantages

- Combines strengths of Uniform Cost Search and Greedy Best-First Search.
- Provides optimal solution if heuristic is admissible.
- Flexible, can be adapted to many domains.

Limitations

- Requires large memory for complex problems.
- Efficiency depends heavily on heuristic function.
- May be slow for very large search spaces.

Applications

- Pathfinding in games (finding shortest route for characters).
- Robotics navigation (robot path planning).
- Maps and GPS systems (finding shortest/fastest route).
- Puzzle solving (8-puzzle, 15-puzzle).
- Network routing (optimal data transfer paths).

Algorithm (Pseudocode)

```
A*(start, goal):
   open_list ← {start}
   closed_list ← {}
   g(start) ← 0
   f(start) ← h(start)

   while open_list is not empty:
      current ← node in open_list with lowest f
      if current = goal:
         return path from start to goal
```

```
remove current from open_list
add current to closed_list

for each neighbor of current:
    tentative_g ← g(current) + cost(current, neighbor)

    if neighbor in closed_list and tentative_g ≥ g(neighbor):
        continue

    if neighbor not in open_list or tentative_g < g(neighbor):
        g(neighbor) ← tentative_g
        f(neighbor) ← g(neighbor) + h(neighbor)
        parent(neighbor) ← current
        if neighbor not in open_list:
            add neighbor to open_list
```

Conclusion

The A* algorithm is one of the most powerful and widely used search algorithms in Artificial Intelligence. By combining actual cost and heuristic estimates, it efficiently finds optimal paths in complex problems. Although memory-intensive, its effectiveness in applications like navigation, robotics, puzzle-solving, and network routing makes it a cornerstone algorithm in AI.