Experiment No: 5

Name:- Winster Pereira
Class:- TE COMPS – A
Roll No:- 9569

Title: Eight puzzle game solution by A* algorithm.

Postlab:

1. Explain the Time Complexity of the A* Algorithm.
Ans: The time complexity of the A* algorithm depends on the characteristics of the search space and the quality of the heuristic function used. In the worst case, the algorithm may need to explore the entire search space, resulting in a time complexity of O(b^d), where b is the branching factor of the search tree and d is the depth of the goal state.
However, if the heuristic function is admissible (i.e., it never overestimates the cost to the goal) and the search space is uniform, the time complexity can be significantly better. In such cases, the time complexity of A* is O(d * log b), where d is the depth of the goal state and b is the branching factor.
The key factor that affects the time complexity is the quality of the heuristic function. A good heuristic function can significantly reduce the number of nodes explored by the algorithm, leading to a faster search process.

2. What are the limitations of A* Algorithm?
Ans: The limitations of A* Algorithm are:
a) Memory requirements:
The A* algorithm needs to store all the nodes in the frontier (open list) and the explored set (closed list), which can consume a significant amount of memory, especially for large search spaces.
b) Sensitivity to heuristic function:
The performance of the A* algorithm heavily depends on the quality of the heuristic function. If the heuristic function is not admissible or does not provide a good estimate of the cost to the goal, the algorithm may not find the optimal solution or may take longer to converge.
c) Difficulty in handling dynamic environments:
The A* algorithm is designed for static environments, where the search space and the heuristic function do not change during the search process. In dynamic environments, where the search space or the heuristic function can change, the A* algorithm may not be able to adapt efficiently.
d) Difficulty in handling complex constraints:
The A* algorithm may struggle to handle complex constraints or multiple objectives in the search problem, as it is primarily designed to optimize a single objective function.

3. Discuss A*, BFS, DFS and Dijkstra’s algorithm in detail with examples.
Ans: The different algorithms are:

a) A* Algorithm:

- A* is an informed search algorithm that uses a heuristic function to guide the search towards the goal.
- It finds the shortest path between the start and goal states by minimizing the sum of the actual cost (g(n)) and the estimated cost (h(n)) to the goal.
- A* is optimal and complete if the heuristic function is admissible (never overestimates the actual cost).
- Example: Solving the 8-puzzle problem using A* with different heuristics.

b) Breadth-First Search (BFS):

- BFS is an uninformed search algorithm that explores all the neighbor nodes at the present depth before moving on to the nodes at the next depth level.
- It guarantees to find the shortest path to the goal, if it exists, but can be memory-intensive for large search spaces.
- BFS is complete and optimal for unweighted graphs, but not necessarily optimal for weighted graphs.
- Example: Finding the shortest path in an unweighted maze using BFS.

c) Depth-First Search (DFS):

- DFS is an uninformed search algorithm that explores as far as possible along each branch before backtracking.
- It may find a solution quickly, but it does not guarantee to find the shortest path.
- DFS is complete for finite search spaces, but it may not terminate for infinite search spaces.
- Example: Solving a maze using DFS, which may find a solution but not necessarily the shortest path.

d) Dijkstra's Algorithm:

- Dijkstra's algorithm is an informed search algorithm that finds the shortest path between two nodes in a weighted graph.
- It always selects the node with the smallest known distance from the set of unvisited nodes.
- Dijkstra's algorithm is optimal for finding the shortest path in a weighted graph, but it may not be efficient for large search spaces.
- Example: Finding the shortest route between two cities in a transportation network using Dijkstra's algorithm.