Fr. Conceicao Rodrigues College of Engineering Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

Department of Computer Engineering

Academic Term II: 23-24

Class: B.E (Computer), Sem – VI Subject Name: Artificial Intelligence Student

Name: Sumit Sanjay Rai **Roll No: 9570**

Practical No:	**5**
Title:	Eight puzzle game solution by A* algorithm
Date of Performance:	**04/03/2024**
Date of Submission:	**11/03/2024**

Rubrics for Evaluation:

Sr. No	**Performance Indicator**	**Excellent**	**Good**	**Below Average**	**Marks**
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Logic/Algorithm Complexity analysis (03)	03(Correct)	02(Partial)	01 (Tried)	
3	Coding Standards (03): Comments/indention/Naming conventions Test Cases /Output	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Assignment (03)	03(done well)	2 (Partially Correct)	1(submitted)	
Total					

Signature of the Teacher:

Post Lab Assignment:

1. Explain the Time Complexity of the A* Algorithms.
2. What are the limitations of A* Algorithms?
3. Discuss A*, BFS, DFS and Dijkstra’s algorithm in detail with examples.

Name: Sumit Sanjay Rai
Roll no: 9570
Class: TE COMPS A.

PAGE NO.:
DATE: / /

PostLab:- Experiment-5.

Q.1. Explain the Time Complexity of the A* Algorithms.

Ans. The time complexity of the A* algorithm depends on the heuristic function's accuracy and the search space's size. In the worst-case scenario, it can be exponential. However, with an admissible and consistent heuristic, A* guarantees finding the optimal solution efficiently.

Q.2. What are the limitations of A* algorithms?

Ans. 1. Memory Usage: A* can consume significant memory, especially in large search spaces.
2. Exponential Complexity: Worst-case time complexity can be exponential, particulorly with ineffective heuristics or large search spaces.
3. Heuristic Dependency: The quality of the heuristic heavily influences A*'s efficieny and optimally.
4. Optimality Assurance: While optimal under certain conditions, A* may not always find the optimal solution.
5. Pathological Cases: A* may encounter scenarios where it explores large portions of the search space inefficiently.
6. Challenges in Dynamic Environments: Adapting A* to dynamic environments can be complex and may require additional techniques.

Q.3. Discuss A*, BFS, DFS and Dijkstra's algorithm in detail with examples.

Ans. ① A* Algorithm:-
Description: A* is a widely used informed search algorithm that finds the shortest path from a start node to a goal node in a graph. It combines the advantages of Dijkstra's algorithm and greedy best-first search by using both the cost to reach a node (g-value) and an estimated cost to reach the goal from the node. (h-value).

Algorithm: 1.1. Initialize an open list and add the start node.

1.2. While the open list is not empty:

- Select the node with the lowest total cost (f-value = g-value + h-value).
- If the selected node is the goal, terminate with success.
- Expand the selected node and add its successors to the open list.

1.3. If the open list becomes empty without reaching the goal, terminate with failure.

Example: Finding the shortest path in a map from city A to city B, where the cost is the distance between cities and the heuristic is the straight-line distance between cities.

② Breadth-First-Search (BFS):

- Description: BFS is an uninformed search algorithm that systematically explores all neighbor nodes at the present depth before moving on to nodes at the next depth level. It guarantees finding the shortest path is an unweighted graph.
- Algorithm: 2.1. Start with the initial node and enqueue it in a queue.

2.2. While the queue is not empty:

- Dequeue a node from the queue.
- If the dequeued node is the goal, terminate with success.
- Enqueue all unvisited neighbor nodes of the dequeued node.

2.3. If the queue becomes empty without reaching the goal, terminate with failure.

- Examples: Exploring all possible move in a maze to find the shortest path from the start to the exit.

③ Depth-first Search (DFS):

- Description: DFS is an uninformed search algorithm that explores as far as possible along each branch before backtracking. It does not guarantee finding the shortest path and can get stuck in deep branches.
- Algorithm: 3.1. Start with the initial node and push it onto a stack.

3.2. While the stack is not empty:

- Pop a node from the stack.
- If the popped node is the goal, terminate with success.
- Push all unvisited neighbor nodes of the popped node onto the stack.

3.3. If the stack becomes empty without reaching the goal, terminate with failure.

- Example: Searching for a specific file in a directory structure by exploring each subdirectory recursively.

④ Dijkstra's Algorithm

- Description: Dijkstra's algorithm is a popular shortest-path algorithm that finds the shortest path from a start node to all other nodes in a weighted graph. It uses a priority queue to select the node with the smallest known distance and updates the distances of its neighbors accordingly.
- Algorithm: 4.1. Initialize all nodes with infinite distance and the start node with distance 0.

4.2. While there are unvisited nodes:

- Select the node with the smallest known distance.
- Update the distances of its neighbors if a shorter path is found.

4.3. Terminate when all nodes have been visited.

- Example: Finding the shortest route for a delivery truck to visit all customers in a city, where distances between locations represent road lengths.