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: **Roll No:**

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:

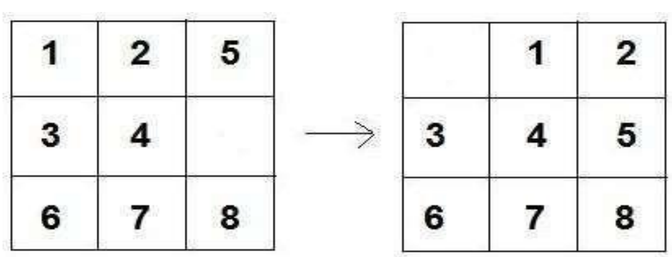
Experiment No: 5

Title: Eight puzzle game solution by A* algorithm

Objective: To study A* algorithm and solutions to 8 puzzle problem using A*

Theory:

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It has a set of 3x3 boards having 9 block spaces out of which, 8 blocks are having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. It has to arrange the tiles in a sequence.
The start state is any situation of tiles, and goal state is tiles arranged in a specific sequence. Solution of this problem is reporting of “movement of tiles” in order to reach the goal state. The transition function or legal move is any one tile movement by one space in any direction (i.e., towards left or right or up or down) if that space is blank.



initial state   goal state

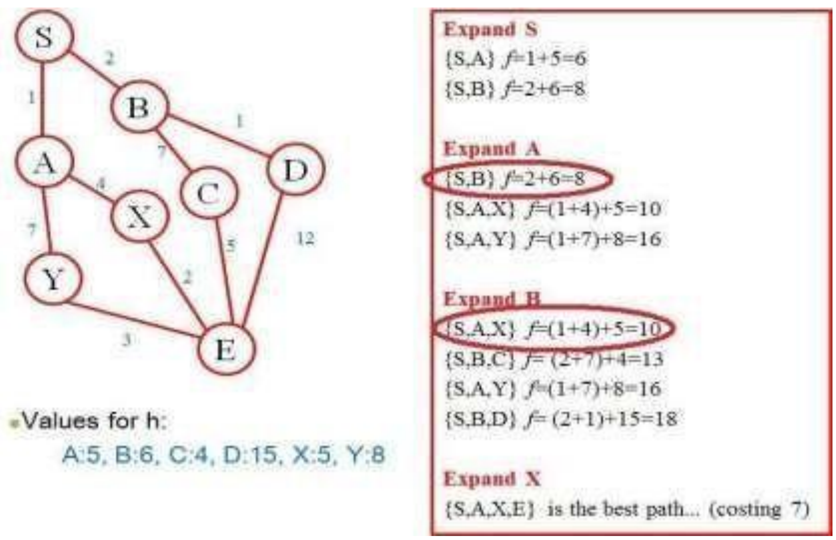
Here the data structure to represent the states can be a 9-element vector indicating the tiles in each board position. Hence, a starting state corresponding to the above configuration will be {1, blank, 4, 6, 5, 8, 2, 3, 7} (there can be various different start positions). The goal state is {1, 2, 3, 4, 5, 6, 7, 8, blank}. Here, the possible movement outcomes after applying a move can be many. They are represented as trees. This tree is called a state space tree. The depth of the tree will depend upon the number of steps in the solution. The part of state space tree of 8-puzzle is shown:

Algorithm:

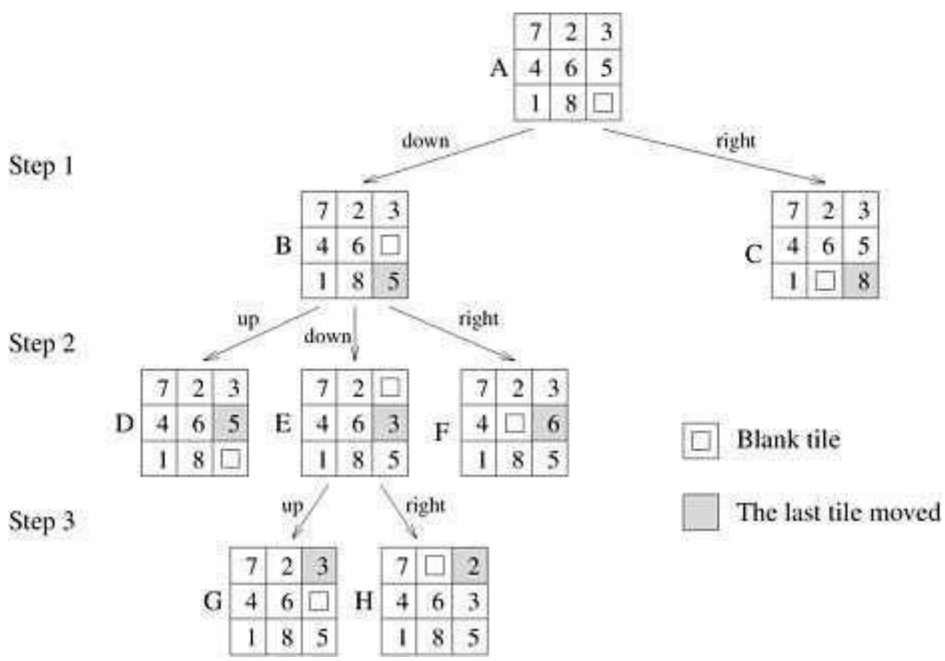
1. **function** A-STAR-SEARCH (initialState, goalTest)
2. return **SUCCESS** or **FAILURE**: /* Cost f(n) = g(n)+h(n)*/
3. frontier = Heap. New(initialState)
4. explored = Set.new()
5. **while not** frontier. isEmpty ();
    a. state = frontier.deleteMin()
    b. explored.add(state)
6. **if** goalTest(state):
    a. return **SUCCESS** (state)
7. **for** neighbor **in** state.neighbours():
    a. **if** neighbor **not in** frontier U explored:
        i. frontier.insert(neighbour)
    b. **else if** neighbor **in** frontier:
        i. frontier.decreaseKey(neighbour)
8. return **FAILURE**

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. It gives the process of plotting an efficiently directed path between multiple points, called nodes. It enjoys widespread use due to its performance and accuracy.

Following is an example of A*



Example 1



Using A* to solve the 8-puzzle problem in a heuristic manner:

- Heuristic 1 (H1): Count the out-of-place tiles, as compared to the goal.
- Heuristic 2 (H2): Sum the distances by which each tile is out of place.

- Heuristic 3 (H3): Multiply the number of required tile reversals by 2.

Analysis of the Evaluation Function:

In developing a good evaluation function for the states in a search space, you are interested in two things:

- g(n) : How far is state n from the start state?
- h(n) : How far is state n from a goal state?

The first value of, g(n), is important because you often want to find the shortest path. This value can be exactly measured by incorporating a **deep count** into the search algorithm.

The second value, h(n), is important for guiding the search towards the goal. It is an **estimated value** based on your heuristic rules.

Evaluation Function: This gives us the following:

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

Post Lab Assignment:

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

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

9526 TE COMPS A A.I Exp5

Postlab:-

① Explain the Time Complexity of the A* algorithm.
=> 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

② What are the limitations of A* algorithms?
=> 1. Memory usage: A* can consume significant memory especially in large search spaces.
2. Exponential Complexity: Worst-case time complexity can be exponential, particularly with ineffective heuristics or large search spaces.
3. Heuristic Dependency: The quality of the heuristic heavily influences A*'s efficiency 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.

⇒ ① A* algorithm:-

• Description: A* is a widely used informed search algorithms 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 uniformed search algorithm that systematically explores all neighbor nodes at the present depth before moving on to nodes at the next depth level. It guarantee finding the shortest path is an unwaighted graph.

Algorithm: 2.1 Start with the initial node and enqueue it in a queue

2.2. Write the queue is not empty:

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

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

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 neighbour nodes

3.3. If the stack becomes empty without reaching the goal

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

④ Dijkstra's Algorithm

- Description: Dijkstra's algo is a popular shortest-path algo that finds the shortest path from a short node to all other nodes in a weighted graph.
- Algorithm: 4.1. Initialize all nodes with infinite distance and the start node with distance.

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.

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