Department of Computer Engineering
Academic Term II: 23-24

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

Student Name: Jatin Kadu **Roll No:9548**

Practical No:	**5**
Title:	Eight puzzle game solution by A* algorithm
Date of Performance:	**4/3/24**
Date of Submission:	**11/3/24**

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:

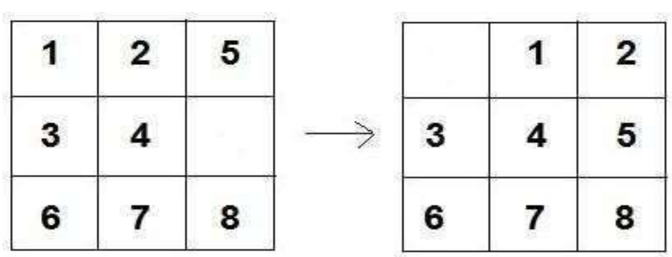
Experiment No: 5

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

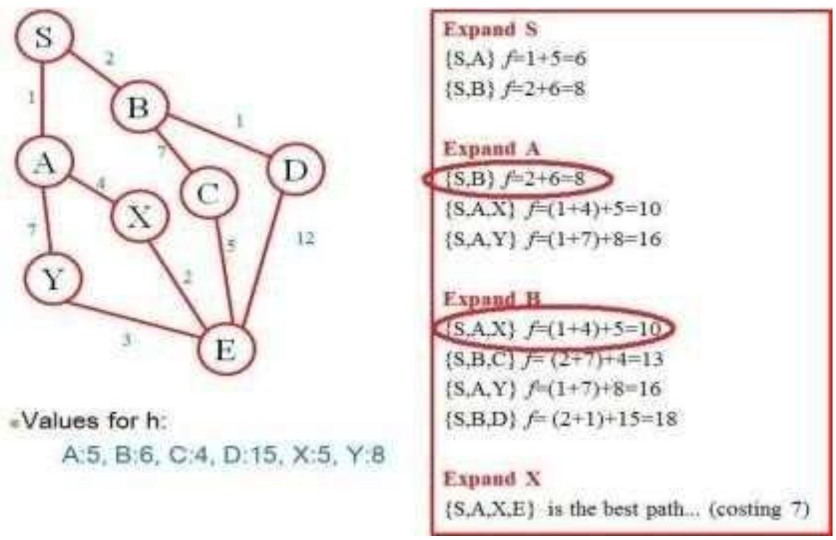
Here the data structure to represent the states can be a 9-element vector indicating the tiles in each board position. Hence, a starting state corresponding to the above configuration will be {1, blank, 4, 6, 5, 8, 2, 3, 7} (there can be various different start positions). The goal state is {1, 2, 3, 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:

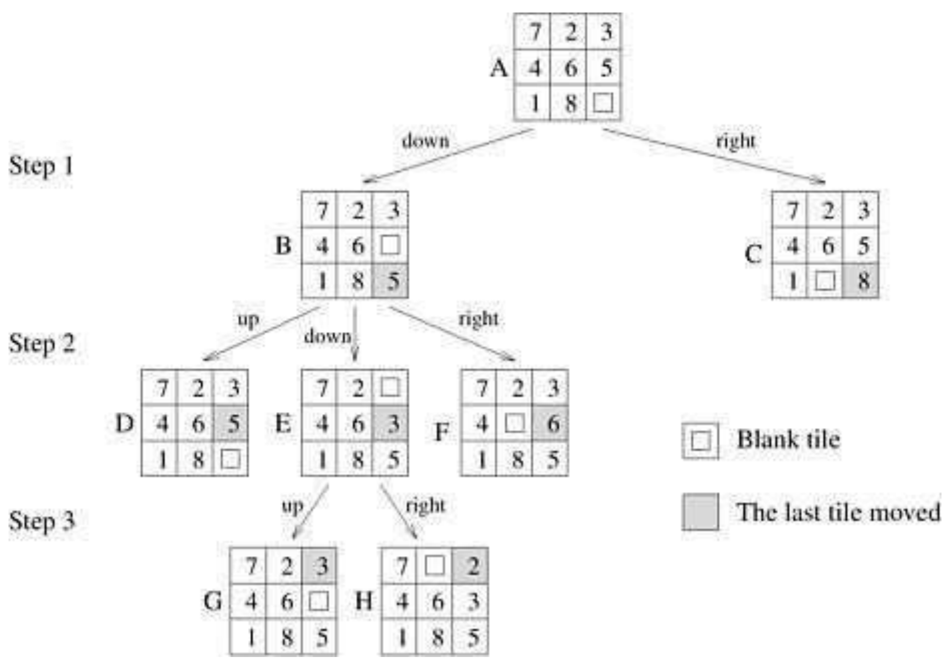
1. **function** A-STAR-SEARCH (initialState, goalTest)
2. return **SUCCESS** or **FAILURE**: /* Cost f(n) = g(n)+h(n)*/
3. frontier = Heap. New(initialState)
4. explored = Set.new()
5. **while not** frontier. isEmpty ();
    a. state = frontier.deleteMin()
    b. explored.add(state)
6. **if** goalTest(state):
    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.

Code:

```
from heapq import heappush, heappop

# Define the goal state for the 8 puzzle problem
GOAL_STATE = (1, 2, 3, 4, 5, 6, 7, 8, 0)  # 0 represents the empty space


class PuzzleState:
    def __init__(self, board, parent=None, cost=0):
        self.board = tuple(board)
        self.parent = parent
        self.cost = cost

    def __lt__(self, other):
        return (self.cost + self.heuristic()) < (other.cost + other.heuristic())

    def __eq__(self, other):
        return self.board == other.board

    def __hash__(self):
        return hash(self.board)

    def heuristic(self):
        # Manhattan distance heuristic
        distance = 0
        for i in range(3):
            for j in range(3):
                if self.board[i * 3 + j] != 0:
                    value = self.board[i * 3 + j] - 1
                    distance += abs(i - (value // 3)) + abs(j - (value % 3))
        return distance

    def is_goal(self):
        return self.board == GOAL_STATE

    def successors(self):
        successors = []
        zero_index = self.board.index(0)
        row, col = zero_index // 3, zero_index % 3

        for dr, dc in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
```

```
            new_row, new_col = row + dr, col + dc
            if 0 <= new_row < 3 and 0 <= new_col < 3:
                new_board = list(self.board)
                new_board[row * 3 + col], new_board[new_row * 3 + new_col] = new_board[new_row * 3 +
new_col], 0
                successors.append(PuzzleState(new_board, parent=self, cost=self.cost + 1))

        return successors


def a_star_search(initial_state):
    frontier = []
    explored = set()

    heappush(frontier, initial_state)

    while frontier:
        current_state = heappop(frontier)

        if current_state.is_goal():
            return current_state

        explored.add(current_state)

        for neighbor in current_state.successors():
            if neighbor not in frontier and neighbor not in explored:
                heappush(frontier, neighbor)
            elif neighbor in frontier:
                existing_neighbor = frontier[frontier.index(neighbor)]
                if neighbor.cost < existing_neighbor.cost:
                    frontier.remove(existing_neighbor)
                    heappush(frontier, neighbor)

    return None  # No solution found


def print_solution(solution_state):
    path = []
    current_state = solution_state
    while current_state:
        path.append(current_state.board)
        current_state = current_state.parent
```

```
    path.reverse()

    for i, state in enumerate(path):
        print(f"Step {i}:")
        print_board(state)
        print()


def print_board(board):
    for i in range(3):
        print(" ".join(str(board[i * 3 + j]) for j in range(3)))


def main():
    # Example initial state
    initial_state = PuzzleState([1, 2, 3, 4, 0, 5, 6, 7, 8])

    solution_state = a_star_search(initial_state)

    if solution_state:
        print("Solution found:")
        print_solution(solution_state)
    else:
        print("No solution found.")


if __name__ == "__main__":
    main()
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SEARCH ERROR                Python

PS C:\Users\hacke\OneDrive\Desktop\SEM VI\AI> & C:/Users/hacke/AppData/Local/Programs/Python/Python311/py
thon.exe "c:/Users/hacke/OneDrive/Desktop/SEM VI/AI/eightpuzzle.py"
Solution found:
Step 0:
1 2 3
4 0 5
6 7 8

Step 1:
1 2 3
4 5 0
6 7 8

Step 2:
1 2 3
4 5 8
6 7 0

Step 3:
1 2 3
4 5 8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR Python

```
Step 4:
1 2 3
4 5 8
0 6 7

Step 5:
1 2 3
0 5 8
4 6 7

Step 6:
1 2 3
5 0 8
4 6 7

Step 7:
1 2 3
5 6 8
4 0 7

Step 8:
1 2 3
5 6 8
4 7 0
```

```
Step 10:
1 2 3
5 0 6
4 7 8

Step 11:
1 2 3
0 5 6
4 7 8

Step 12:
1 2 3
4 5 6
0 7 8

Step 13:
1 2 3
4 5 6
7 0 8

Step 14:
1 2 3
4 5 6
7 8 0
```