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: Nimish Ravindra Patil **Roll No: 9565**

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:

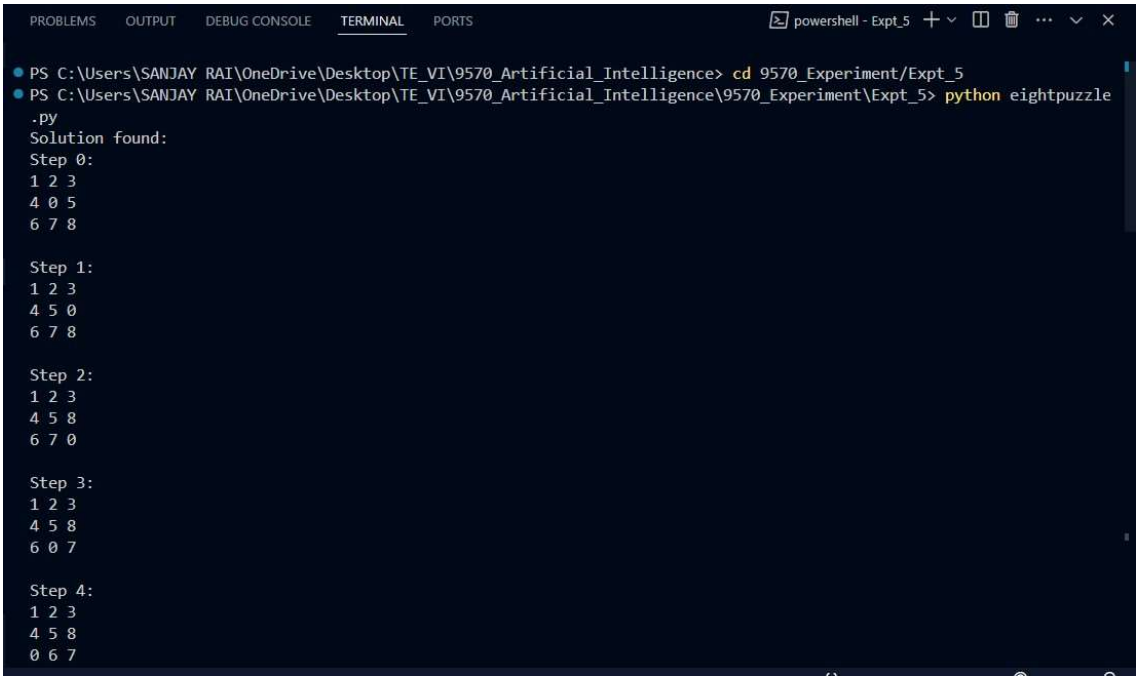
Source code: from heapq import heappush, heappop

```
# 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:



```
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
```