A* ALGORITHM

IMPLEMENTATION

```
print("1BM22CS324")
print("V DHANUSH REDDY")
from collections import deque
import heapq

def solve_8puzzle_astar(initial_state):
    """
    Solves the 8-puzzle using A* Search with the misplaced tiles heuristic.

    Args:
        initial_state: A list of lists representing the initial state of the puzzle.

    Returns:
        A list of lists representing the solution path, or None if no solution is found.
    """

    def find_blank(state):
        """Finds the row and column of the blank tile (0)."""
        for row in range(3):
            for col in range(3):
                if state[row][col] == 0:
                    return row, col

    def get_neighbors(state):
        """Generates possible neighbor states by moving the blank tile."""
        row, col = find_blank(state)
```

```
        neighbors = []

        # Possible moves: Up, Down, Left, Right
        if row > 0:  # Up
            new_state = [r[:] for r in state]
            new_state[row][col], new_state[row - 1][col] = new_state[row - 1][col],
new_state[row][col]
            neighbors.append(new_state)
        if row < 2:  # Down
            new_state = [r[:] for r in state]
            new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col],
new_state[row][col]
            neighbors.append(new_state)
        if col > 0:  # Left
            new_state = [r[:] for r in state]
            new_state[row][col], new_state[row][col - 1] = new_state[row][col - 1],
new_state[row][col]
            neighbors.append(new_state)
        if col < 2:  # Right
            new_state = [r[:] for r in state]
            new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1],
new_state[row][col]
            neighbors.append(new_state)

        return neighbors

    def misplaced_tiles_heuristic(state):
        """Calculates the number of misplaced tiles."""
        misplaced_count = 0
        for row in range(3):
            for col in range(3):
                if state[row][col] != 0 and state[row][col] != (row * 3 + col + 1):
```

```
            misplaced_count += 1
    return misplaced_count

goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Print initial and goal states
print("Initial State:")
for row in initial_state:
    print(row)
print("\nGoal State:")
for row in goal_state:
    print(row)
print("\nStarting A* Search...\n")

priority_queue = [(misplaced_tiles_heuristic(initial_state), 0, initial_state, [])]
visited = set()

while priority_queue:
    _, cost, current_state, path = heapq.heappop(priority_queue)

    # Check if the goal state is reached
    if current_state == goal_state:
        return path + [current_state]

    # Mark the current state as visited
    visited.add(tuple(map(tuple, current_state)))

    # Explore neighbors
    for neighbor in get_neighbors(current_state):
        if tuple(map(tuple, neighbor)) not in visited:
```

```
            new_cost = cost + 1
            heuristic_cost = misplaced_tiles_heuristic(neighbor)
            heapq.heappush(priority_queue, (new_cost + heuristic_cost, new_cost, neighbor,
path + [current_state]))

    return None  # No solution found

# Example usage:
initial_state = [[1, 2, 3], [4, 0, 6], [7, 5, 8]]
solution = solve_8puzzle_astar(initial_state)
if solution:
    print("\nSolution found:")
    for state in solution:
        for row in state:
            print(row)
        print()
else:
    print("No solution found.")
```

OUTPUT:

```
1BM22CS324
V DHANUSH REDDY
Initial State:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

Goal State:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Starting BFS...


Solution found:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

OBSERVATION:

25-10-24 | Week-3 |

Q) A* Algorithm

→ function A* search(problem) returns a solution or failure

node ← a node n with n state= problem.initial stage, neg=0

frontier ← a priority queue ordered by ascending g+h, only element

loop do
  if empty ? (frontier) then return failure
  n ← pop(frontier)
  if problem.goalTest(n.state) then
    return solution(n)
  for each action a in problem.action(n.state) do
    n' ← childNode(problem, n, a)
    insert(n', g(n') + h(n'), frontier)

→ $f(n) = g(n) + h(n)$

$f(n)$ → Evaluation cost function which gives the cheapest solution cost.

$g(n)$ → Exact cost to reach node n from initial state.

$h(n)$ → Estimation of assumed cost from current state (n) to reach goal.

$h(n)$ → no of misplaced tiles.

$h(n)$ → Manhatton Distance.

Manhattan Distance

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix}$$

initial

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 7 & 8 \end{bmatrix} f(n)=2 \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

initial — Goal.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 4 & 7 & 8 \end{bmatrix} f(n)=4 \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 8 \end{bmatrix} f(n)=1$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 6 \\ 7 & 5 & 8 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

$f(n)=2$ — goal state.

25/10

1) Misplaced tiles.

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix} \quad \begin{array}{l} g(n)=0 \\ h(n)=4 \\ f(n)=4. \end{array} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

initial   goal.

$g(n)=1$

$$\begin{array}{l} h(n)=5 \\ f(n)=6 \end{array} \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 5 & 0 \end{bmatrix} \qquad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 0 & 7 & 5 \end{bmatrix} \begin{array}{l} h(n)=5 \\ f(n)=6 \end{array} \qquad \begin{array}{l} h(n)=3 \\ f(n)=4 \end{array} \begin{bmatrix} 2 & 8 & 3 \\ 1 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$g(n)=2$

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 4 & 0 \\ 7 & 6 & 5 \end{bmatrix} \qquad \begin{bmatrix} 2 & 0 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \qquad \begin{bmatrix} 2 & 8 & 3 \\ 0 & 1 & 4 \\ 7 & 6 & 5 \end{bmatrix} \qquad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix}$$

$f(n)=6$ $f(n)=5$ $f(n)=5$ $f(n)=6$

$g(n)=3$

$$\begin{bmatrix} 0 & 2 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} f(n)=5 \qquad \begin{bmatrix} 2 & 3 & 0 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} f(n)=7$$

$g(n)=4$

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

solution found

$f(n)=5$