**✅ Python Program: 8-Puzzle Solver using A\* Search**

**PROGRAM:**

import heapq

# Define goal state

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]] # 0 represents the blank tile

# Directions: Up, Down, Left, Right

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# Node class

class Node:

def \_\_init\_\_(self, state, parent=None, move="", cost=0):

self.state = state

self.parent = parent

self.move = move

self.cost = cost

self.priority = cost + self.heuristic()

def \_\_lt\_\_(self, other):

return self.priority < other.priority

def heuristic(self):

"""Manhattan Distance"""

distance = 0

for i in range(3):

for j in range(3):

value = self.state[i][j]

if value != 0:

goal\_i = (value - 1) // 3

goal\_j = (value - 1) % 3

distance += abs(i - goal\_i) + abs(j - goal\_j)

return distance

def find\_blank(self):

for i in range(3):

for j in range(3):

if self.state[i][j] == 0:

return i, j

def get\_neighbors(self):

neighbors = []

x, y = self.find\_blank()

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

new\_state = [row[:] for row in self.state]

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

move = f"Move {new\_state[x][y]} {'Up' if dx==-1 else 'Down' if dx==1 else 'Left' if dy==-1 else 'Right'}"

neighbors.append(Node(new\_state, self, move, self.cost + 1))

return neighbors

def is\_goal(self):

return self.state == goal\_state

def print\_path(self):

path = []

node = self

while node:

path.append((node.move, node.state))

node = node.parent

path.reverse()

for move, state in path:

if move:

print(f"\n{move}")

for row in state:

print(row)

def a\_star(start\_state):

start\_node = Node(start\_state)

open\_list = []

heapq.heappush(open\_list, start\_node)

visited = set()

while open\_list:

current\_node = heapq.heappop(open\_list)

if current\_node.is\_goal():

print("\n✅ Solution found:")

current\_node.print\_path()

return

visited.add(str(current\_node.state))

for neighbor in current\_node.get\_neighbors():

if str(neighbor.state) not in visited:

heapq.heappush(open\_list, neighbor)

print("❌ No solution found.")

# Example start state

start = [[1, 2, 3],

[5, 0, 6],

[4, 7, 8]]

a\_star(start)

