**PROGRAM:**

import heapq

def get\_manhattan\_distance(board, goal):

distance = 0

for i in range(3):

for j in range(3):

if board[i][j] != 0:

x, y = divmod(goal.index(board[i][j]), 3)

distance += abs(x - i) + abs(y - j)

return distance

def get\_neighbors(board):

neighbors = []

x, y = [(i, j) for i in range(3) for j in range(3) if board[i][j] == 0][0]

moves = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]

for i, j in moves:

if 0 <= i < 3 and 0 <= j < 3:

new\_board = [row[:] for row in board]

new\_board[x][y], new\_board[i][j] = new\_board[i][j], new\_board[x][y]

neighbors.append(new\_board)

return neighbors

def best\_first\_search(initial, goal):

initial\_flat = [num for row in initial for num in row]

goal\_flat = [num for row in goal for num in row]

heap = [(get\_manhattan\_distance(initial, goal\_flat), 0, initial, [initial])]

visited = set()

visited.add(tuple(tuple(row) for row in initial))

while heap:

\_, moves, current, path = heapq.heappop(heap)

if current == goal:

return path

for neighbor in get\_neighbors(current):

neighbor\_tuple = tuple(tuple(row) for row in neighbor)

if neighbor\_tuple not in visited:

visited.add(neighbor\_tuple)

heapq.heappush(heap, (get\_manhattan\_distance(neighbor, goal\_flat) + moves + 1, moves + 1, neighbor, path + [neighbor]))

return []

def get\_input\_board(prompt):

print(prompt)

board = []

for \_ in range(3):

row = list(map(int, input().strip().split()))

board.append(row)

return board

def print\_board(board):

for row in board:

print(" ".join(map(str, row)))

print()

initial\_board = get\_input\_board("Enter the initial board (3x3) row-wise (use 0 for the blank space):")

goal\_board = get\_input\_board("Enter the goal board (3x3) row-wise (use 0 for the blank space):")

path = best\_first\_search(initial\_board, goal\_board)

if path:

print(f"Solved in {len(path) - 1} moves!")

print("Intermediate steps:")

for step in path:

print\_board(step)

else:

print("No solution found.")

**OUTPUT:**



