8 puzzle

import heapq

GOAL\_STATE = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

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

def manhattan\_distance(state):

distance = 0

for i in range(3):

for j in range(3):

val = state[i][j]

if val != 0:

goal\_x, goal\_y = (val - 1) // 3, (val - 1) % 3

distance += abs(i - goal\_x) + abs(j - goal\_y)

return distance

def find\_zero(state):

for i in range(3):

for j in range(3):

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

return i, j

def is\_goal(state):

return state == GOAL\_STATE

def state\_to\_tuple(state):

return tuple(tuple(row) for row in state)

def get\_neighbors(state):

neighbors = []

x, y = find\_zero(state)

for dx, dy in MOVES:

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < 3 and 0 <= new\_y < 3:

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

new\_state[x][y], new\_state[new\_x][new\_y] = new\_state[new\_x][new\_y], new\_state[x][y]

neighbors.append(new\_state)

return neighbors

def a\_star(start\_state):

open\_set = []

heapq.heappush(open\_set, (manhattan\_distance(start\_state), 0, start\_state))

came\_from = {}

g\_score = {state\_to\_tuple(start\_state): 0}

while open\_set:

\_, cost, current = heapq.heappop(open\_set)

if is\_goal(current):

path = []

while state\_to\_tuple(current) in came\_from:

path.append(current)

current = came\_from[state\_to\_tuple(current)]

path.append(start\_state)

return list(reversed(path))

for neighbor in get\_neighbors(current):

neighbor\_tuple = state\_to\_tuple(neighbor)

tentative\_g = g\_score[state\_to\_tuple(current)] + 1

if neighbor\_tuple not in g\_score or tentative\_g < g\_score[neighbor\_tuple]:

came\_from[neighbor\_tuple] = current

g\_score[neighbor\_tuple] = tentative\_g

f\_score = tentative\_g + manhattan\_distance(neighbor)

heapq.heappush(open\_set, (f\_score, tentative\_g, neighbor))

return None

def print\_state(state):

for row in state:

print(' '.join(str(num) if num != 0 else '\_' for num in row))

print()

start = [[1, 2, 3],

[4, 0, 6],

[7, 5, 8]]

solution = a\_star(start)

if solution:

print("Solution found in", len(solution)-1, "moves:\n")

for step in solution:

print\_state(step)

else:

print("No solution found.")

**output**

