**8 SQUARE PUZZLE**

import heapq

def is\_valid\_input(matrix):

flat\_matrix = [char for row in matrix for char in row]

if len(flat\_matrix) != 9:

return False, "The matrix must contain exactly 9 characters."

blanks = {' '}

non\_blanks = [char for char in flat\_matrix if char not in blanks]

if len(set(non\_blanks)) != len(non\_blanks):

return False, "Invalid input. Please ensure no duplicates, except for the blank space."

return True, ""

def get\_matrix\_input(prompt):

while True:

print(prompt)

input\_str = input().strip()

flat\_input = [char.strip() if char.strip() else ' ' for char in input\_str.split(",")]

if len(flat\_input) != 9:

print("Invalid input. Please enter exactly 9 valid characters (use 1-8 numbers and symbols and ' ' for blank space).")

continue

matrix = [flat\_input[i:i + 3] for i in range(0, 9, 3)]

valid, message = is\_valid\_input(matrix)

if not valid:

print(message)

continue

return matrix

def misplaced\_tiles(state, goal):

return sum(1 for i in range(9) if state[i] != goal[i] and state[i] not in {' ', '0', '\_'})

def print\_chessboard(matrix):

for j in range(3):

print(" | ".join(f"{matrix[k + j \* 3]:2s}" for k in range(3)))

if j < 2:

print("-----------")

s = get\_matrix\_input("Enter Source States (9 characters separated by commas, use ' ' for blank space):")

des = get\_matrix\_input("Enter Destination States (9 characters separated by commas, use ' ' for blank space):")

s = tuple(' ' if char in {' ', '0', '\_'} else char for row in s for char in row)

des = tuple(' ' if char in {' ', '0', '\_'} else char for row in des for char in row)

directions = {0: "up", 1: "down", 2: "left", 3: "right"}

print("Initial State:")

print\_chessboard(s)

pq = []

d = {}

g = {}

d[s] = (s, -1, None)

g[s] = 0

heapq.heappush(pq, (misplaced\_tiles(s, des), s, 0))

f = 0

while pq:

\_, cur, move\_num = heapq.heappop(pq)

if cur == des:

f = 1

break

for i in range(9):

if cur[i] in {' ', '0', '\_'}:

idx = i

break

temp = list(cur)

if idx >= 3:

a = temp[:]

a[idx], a[idx - 3] = a[idx - 3], a[idx]

a = tuple(a)

if a not in g or g[a] > g[cur] + 1:

g[a] = g[cur] + 1

f\_cost = g[a] + misplaced\_tiles(a, des)

heapq.heappush(pq, (f\_cost, a, move\_num + 1))

d[a] = (cur, move\_num + 1, 0)

if idx <= 5:

a = temp[:]

a[idx], a[idx + 3] = a[idx + 3], a[idx]

a = tuple(a)

if a not in g or g[a] > g[cur] + 1:

g[a] = g[cur] + 1

f\_cost = g[a] + misplaced\_tiles(a, des)

heapq.heappush(pq, (f\_cost, a, move\_num + 1))

d[a] = (cur, move\_num + 1, 1)

if idx % 3 != 2:

a = temp[:]

a[idx], a[idx + 1] = a[idx + 1], a[idx]

a = tuple(a)

if a not in g or g[a] > g[cur] + 1:

g[a] = g[cur] + 1

f\_cost = g[a] + misplaced\_tiles(a, des)

heapq.heappush(pq, (f\_cost, a, move\_num + 1))

d[a] = (cur, move\_num + 1, 3)

if idx % 3 != 0:

a = temp[:]

a[idx], a[idx - 1] = a[idx - 1], a[idx]

a = tuple(a)

if a not in g or g[a] > g[cur] + 1:

g[a] = g[cur] + 1

f\_cost = g[a] + misplaced\_tiles(a, des)

heapq.heappush(pq, (f\_cost, a, move\_num + 1))

d[a] = (cur, move\_num + 1, 2)

if f == 0:

print("No solution found")

else:

ans = []

while d[cur][1] != -1:

ans.append((cur, d[cur][1], d[cur][2]))

cur = d[cur][0]

print("Solution found!")

print(f"Total moves to reach the goal: {len(ans)}\n")

ans.reverse()

for step, move\_number, move\_direction in ans:

print(f"Move number: {move\_number}")

print(f"Move direction: {directions[move\_direction]}")

print("States after the move:")

print\_chessboard(step)

g\_x = g[step]

h\_x = misplaced\_tiles(step, des)

f\_x = g\_x + h\_x

print(f"f(x) = g(x) + h(x) = {g\_x} + {h\_x} = {f\_x}")

print("-" \* 10)

print(f"Final state after {len(ans)} moves:")

print\_chessboard(des)