def print\_state(state):

for row in state:

row\_str = ""

for num in row:

row\_str += str(num) + " "

print(row\_str)

print()

# Function to find the blank tile in the puzzle

def find\_blank(state):

for i in range(3):

for j in range(3):

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

return i, j

def move(state,direction):

i,j=find\_blank(state)

new\_state = []

for row in state:

new\_row = row[:]

new\_state.append(new\_row)

if direction=="up":

if i > 0:

new\_state[i][j], new\_state[i-1][j] = new\_state[i-1][j], new\_state[i][j]

return new\_state

else:

return None

if direction=="down":

if i < 2:

new\_state[i][j], new\_state[i+1][j] = new\_state[i+1][j], new\_state[i][j]

return new\_state

else:

return None

if direction=="left":

if j > 0:

new\_state[i][j], new\_state[i][j-1] = new\_state[i][j-1], new\_state[i][j]

return new\_state

else:

return None

if direction=="right":

if j < 2:

new\_state[i][j], new\_state[i][j+1] = new\_state[i][j+1], new\_state[i][j]

return new\_state

else:

return None

def calculate\_heuristic(state, goal\_state):

# Simple heuristic: count the number of misplaced tiles

h = 0

for i in range(3):

for j in range(3):

if state[i][j] != goal\_state[i][j]:

h += 1

return h

def a\_star(initial\_state, goal\_state):

OPEN = [(calculate\_heuristic(initial\_state, goal\_state), 0, initial\_state)]

CLOSED = set()

found=False

while OPEN:

f, g, current\_state = min(OPEN)

OPEN.remove((f, g, current\_state))

CLOSED.add(tuple(map(tuple, current\_state)))

print\_state(current\_state)

if current\_state == goal\_state:

print("Solution found!")

found=True

break

steps=["up","down","left","right"]

successors = [move(current\_state,direction) for direction in steps]

suc = []

for s in successors:

if s:

if (tuple(map(tuple, s)) not in CLOSED):

suc.append(s)

for successor in suc:

h = calculate\_heuristic(successor, goal\_state)

g\_successor = g + 1

f\_successor = g\_successor + h

if (f\_successor, g\_successor, successor) not in OPEN:

OPEN.append((f\_successor, g\_successor, successor))

if found==False:

print("Solution not found")

# Updated example usage with the provided input

initial\_state = [

[1, 2, 3],

[8, 0, 4],

[7, 6, 5]

]

goal\_state = [

[1, 2, 0],

[8, 6, 3],

[7, 5, 4]

]

a\_star(initial\_state, goal\_state)