8 PUZZLE USING DFS AND BFS

from collections import deque

# Function to check if the current state is the goal state

def is\_goal(state, goal\_state):

return state == goal\_state

# Function to get the possible moves from the current state

def get\_neighbors(state):

neighbors = []

index = state.index(0) # Find the blank (0)

row, col = divmod(index, 3)

# Define possible moves (up, down, left, right)

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

for move in moves:

new\_row, new\_col = row + move[0], col + move[1]

if 0 <= new\_row < 3 and 0 <= new\_col < 3:

new\_index = new\_row \* 3 + new\_col

new\_state = list(state)

new\_state[index], new\_state[new\_index] = new\_state[new\_index], new\_state[index]

neighbors.append(tuple(new\_state))

return neighbors

# Function to print the state in a 3x3 format

def print\_state(state):

for i in range(0, 9, 3):

print(state[i:i+3])

print() # Blank line for readability

# DFS algorithm to solve the 8-puzzle

def dfs(start, goal):

visited = set()

stack = [(start, [])]

while stack:

current\_state, path = stack.pop()

if current\_state in visited:

continue

visited.add(current\_state)

if is\_goal(current\_state, goal):

return path + [current\_state]

for neighbor in get\_neighbors(current\_state):

if neighbor not in visited:

stack.append((neighbor, path + [current\_state]))

return None # If no solution found

# Function to take 8-puzzle input from the user

def input\_puzzle(prompt):

print(prompt)

puzzle = []

for i in range(3):

row = input(f"Enter row {i + 1} (3 numbers separated by spaces): ").split()

puzzle.extend([int(x) for x in row])

return tuple(puzzle)

# Function to select goal state

def select\_goal\_state():

print("Select a goal state:")

print("1. Goal State:")

print(" 0 1 2")

print(" 3 4 5")

print(" 6 7 8")

print("2. Goal State:")

print(" 1 2 3")

print(" 4 5 6")

print(" 7 8 0")

choice = input("Enter 1 or 2: ")

if choice == '1':

return (0, 1, 2, 3, 4, 5, 6, 7, 8)

else:

return (1, 2, 3, 4, 5, 6, 7, 8, 0)

# Main code

start\_state = input\_puzzle("Enter the start state (use 0 for the blank space):")

goal\_state = select\_goal\_state()

print("\nSolving using DFS...")

dfs\_solution = dfs(start\_state, goal\_state)

if dfs\_solution:

print("DFS Solution found! Steps:")

for i, step in enumerate(dfs\_solution):

print(f"Step {i + 1}:")

print\_state(step)

else:

print("No solution found using DFS.")

DFS:

from collections import deque

# Function to check if the current state is the goal state

def is\_goal(state, goal\_state):

return state == goal\_state

# Function to get the possible moves from the current state

def get\_neighbors(state):

neighbors = []

index = state.index(0) # Find the blank (0)

row, col = divmod(index, 3)

# Define possible moves (up, down, left, right)

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

for move in moves:

new\_row, new\_col = row + move[0], col + move[1]

if 0 <= new\_row < 3 and 0 <= new\_col < 3:

new\_index = new\_row \* 3 + new\_col

new\_state = list(state)

new\_state[index], new\_state[new\_index] = new\_state[new\_index], new\_state[index]

neighbors.append(tuple(new\_state))

return neighbors

# Function to print the state in a 3x3 format

def print\_state(state):

for i in range(0, 9, 3):

print(state[i:i+3])

print() # Blank line for readability

# DFS algorithm to solve the 8-puzzle

def dfs(start, goal):

visited = set()

stack = [(start, [])]

while stack:

current\_state, path = stack.pop()

if current\_state in visited:

continue

visited.add(current\_state)

if is\_goal(current\_state, goal):

return path + [current\_state]

for neighbor in get\_neighbors(current\_state):

if neighbor not in visited:

stack.append((neighbor, path + [current\_state]))

return None # If no solution found

# Function to take 8-puzzle input from the user

def input\_puzzle(prompt):

print(prompt)

puzzle = []

for i in range(3):

row = input(f"Enter row {i + 1} (3 numbers separated by spaces): ").split()

puzzle.extend([int(x) for x in row])

return tuple(puzzle)

# Function to select goal state

def select\_goal\_state():

print("Select a goal state:")

print("1. Goal State:")

print(" 0 1 2")

print(" 3 4 5")

print(" 6 7 8")

print("2. Goal State:")

print(" 1 2 3")

print(" 4 5 6")

print(" 7 8 0")

choice = input("Enter 1 or 2: ")

if choice == '1':

return (0, 1, 2, 3, 4, 5, 6, 7, 8)

else:

return (1, 2, 3, 4, 5, 6, 7, 8, 0)

# Main code

start\_state = input\_puzzle("Enter the start state (use 0 for the blank space):")

goal\_state = select\_goal\_state()

print("\nSolving using DFS...")

dfs\_solution = dfs(start\_state, goal\_state)

if dfs\_solution:

print("DFS Solution found! Steps:")

for i, step in enumerate(dfs\_solution):

print(f"Step {i + 1}:")

print\_state(step)

else:

print("No solution found using DFS.")

STATE SPACE TREE AND OUTPUT:





