# BFS:

## Code:

def bfs(src,target):

queue = []

queue.append(src)

exp = []

while len(queue) > 0:

source = queue.pop(0)

exp.append(source)

print\_state(source)

if source==target:

print("success")

return

poss\_moves\_to\_do = []

poss\_moves\_to\_do = possible\_moves(source,exp)

for move in poss\_moves\_to\_do:

if move not in exp and move not in queue:

queue.append(move)

def print\_state(state):

for i in range(9):

if i%3==0:

print("\n")

if state[i]==0:

print("\_ ",end="")

else:

print(str(state[i])+" ",end="")

print("\n")

def possible\_moves(state,visited\_states):

#index of empty spot

b = state.index(0)

#directions array

d = []

#Add all the possible directions

if b not in [0,1,2]:

d.append('u')

if b not in [6,7,8]:

d.append('d')

if b not in [0,3,6]:

d.append('l')

if b not in [2,5,8]:

d.append('r')

# If direction is possible then add state to move

pos\_moves\_it\_can = []

# for all possible directions find the state if that move is played

### Jump to gen function to generate all possible moves in the given directions

for i in d:

pos\_moves\_it\_can.append(gen(state,i,b))

return [move\_it\_can for move\_it\_can in pos\_moves\_it\_can if move\_it\_can not in visited\_states]

def gen(state, m, b):

temp = state.copy()

if m=='d':

temp[b+3],temp[b] = temp[b],temp[b+3]

if m=='u':

temp[b-3],temp[b] = temp[b],temp[b-3]

if m=='l':

temp[b-1],temp[b] = temp[b],temp[b-1]

if m=='r':

temp[b+1],temp[b] = temp[b],temp[b+1]

# return new state with tested move to later check if "src == target"

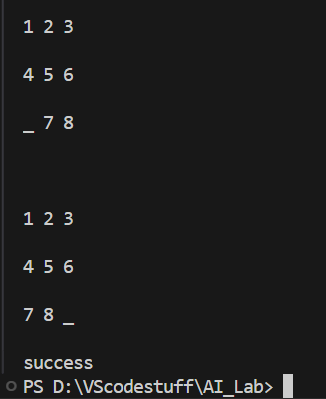
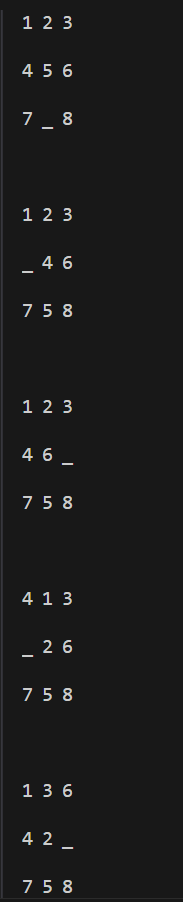
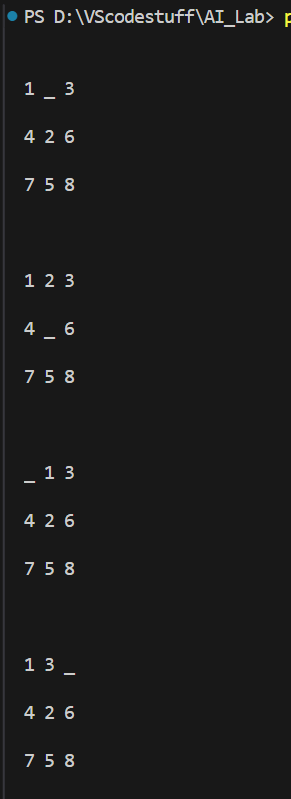
return temp

src = [1,0,3,4,2,6,7,5,8]

target = [1,2,3,4,5,6,7,8,0]

bfs(src, target)

## Output:



# DFS:

## Code:

cnt=0

def print\_state(in\_array):

global cnt

cnt += 1

for row in in\_array:

print(' '.join(str(num) for num in row))

print() # Print a blank line for better readability

def helper(goal, in\_array, row, col, vis):

# Mark the current position as visited

vis[row][col] = 1

drow = [-1, 0, 1, 0] # Directions for row movements: up, right, down, left

dcol = [0, 1, 0, -1] # Directions for column movements

dchange = ['U', 'R', 'D', 'L']

# Print the current state

print("Current state:")

print\_state(in\_array)

# Check if the current state is the goal state

if in\_array == goal:

print\_state(in\_array)

print(f"Number of states : {cnt}")

return True

# Explore all possible directions

for i in range(4):

nrow = row + drow[i]

ncol = col + dcol[i]

# Check if the new position is within bounds and not visited

if 0 <= nrow < len(in\_array) and 0 <= ncol < len(in\_array[0]) and not vis[nrow][ncol]:

# Make the move (swap the empty space with the adjacent tile)

print(f"Took a {dchange[i]} move")

in\_array[row][col], in\_array[nrow][ncol] = in\_array[nrow][ncol], in\_array[row][col]

# Recursive call

if helper(goal, in\_array, nrow, ncol, vis):

return True

# Backtrack (undo the move)

in\_array[row][col], in\_array[nrow][ncol] = in\_array[nrow][ncol], in\_array[row][col]

# Mark the position as unvisited before returning

vis[row][col] = 0

return False

# Example usage

initial\_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]] # 0 represents the empty space

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

visited = [[0] \* 3 for \_ in range(3)] # 3x3 visited matrix

empty\_row, empty\_col = 1, 0 # Initial position of the empty space

found\_solution = helper(goal\_state, initial\_state, empty\_row, empty\_col, visited)

print("Solution found:", found\_solution)

## Output:

