def bfs(src,target):

queue = []

queue.append(src)

exp = []

while len(queue) > 0:

source = queue.pop(0)

exp.append(source)

print(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 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"

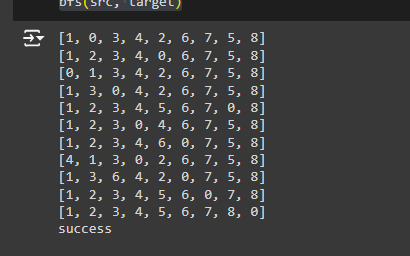
return temp

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:

from collections import deque

def is\_goal(state):

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

def get\_neighbors(state):

neighbors = []

index = state.index(0)

if index > 2:

# Move up

new\_state = state.copy()

new\_state[index], new\_state[index - 3] = new\_state[index - 3], new\_state[index]

neighbors.append(new\_state)

if index < 6:

# Move down

new\_state = state.copy()

new\_state[index], new\_state[index + 3] = new\_state[index + 3], new\_state[index]

neighbors.append(new\_state)

if index % 3 > 0:

# Move left

new\_state = state.copy()

new\_state[index], new\_state[index - 1] = new\_state[index - 1], new\_state[index]

neighbors.append(new\_state)

if index % 3 < 2:

# Move right

new\_state = state.copy()

new\_state[index], new\_state[index + 1] = new\_state[index + 1], new\_state[index]

neighbors.append(new\_state)

return neighbors

def dfs(state):

queue = deque([(state, [state])])

visited = set(tuple(state))

while queue:

node, path = queue.popleft()

if is\_goal(node):

return path

for neighbor in get\_neighbors(node):

if tuple(neighbor) not in visited:

queue.append((neighbor, path + [neighbor]))

visited.add(tuple(neighbor))

return None

# Test the function

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

solution = dfs(initial\_state)

if solution:

print("Solution found:")

for state in solution:

print(state)

else:

print("No solution found")

Output:

