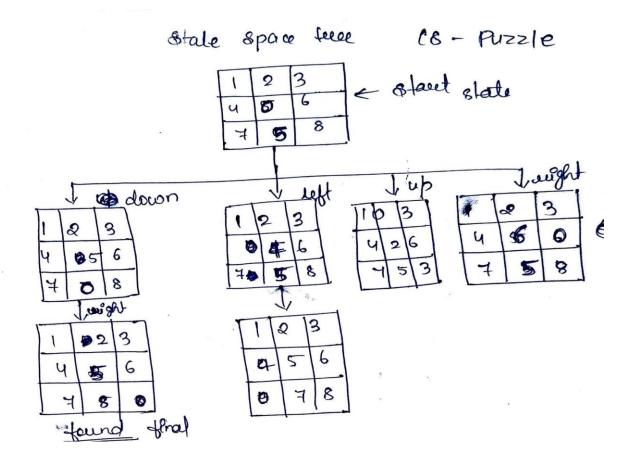
### LAB PROGRAM 2

Solve 8 – Puzzle problems.



# LAB-2

## 8 Puzzle Puoblem

Proposed humby sous no mb det bls(301c, lauget):
"queire =[(501c, None)] # slate and last move visited = Bet()

" to alate courd = 0 # Intralize state court

while queues

C' stati i dastmove = queue. poplo State tuble = tuble (state) # control state to (0,0) . tuple afor set

(miles more visited add (state-tuble)

state \_ court +=1 = 1 novement the state

punt-board (state) of the print power (state) of last move; (state) of point (f" Cultipht move: § l'ast move) (n')

( storm [0.0(0) : - ' clam and status (0.0(1) - - 'd an'

of back spurnetrolly Grain , state ochieved (") bueak

four move, direction in possible-moves

if tuble (move) not en visited:

queue. append ((move, derection))

perint()" Total unique slates explored: § slate\_ count 3")

```
def possible-mouse (slate):

b = slate . Index (0)

disections = []

ef b not en [0,1,2]: disections append ('u')

if b not en [0,3,6]: disections append ('d')

if b not en [0,3,6]: disections append ('l')

if b not en [0,3,6]: disections append ('l')

if b not en [0,5,8]: disections append ('l')

uetroun [(get n (slate, d.b), d) for d in directions]

ded gen (slate disection, b):

ded gen (slate dis
```

maxDepth (= 03

board = np. avvoy (Slate); veshape (3,3);

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

bf3( soic, larget) color the puzzle

(-1 - (0 - ...)

#### **Code:**

```
import numpy as np
def bfs(src, target):
  queue = [(src, None)] # State and last move
  visited = set()
  state_count = 0 # Initialize state count
  while queue:
     state, last_move = queue.pop(0)
     state_tuple = tuple(state) # Convert state to tuple for set operations
     if state_tuple not in visited:
       visited.add(state_tuple)
       state_count += 1 # Increment the state count
       print_board(state)
       if last_move:
          print(f"Current move: {last_move}\n")
       if state == target:
          print("Goal state achieved!")
          break
```

for move, direction in possible\_moves(state):

```
if tuple(move) not in visited:
             queue.append((move, direction))
  print(f"Total unique states explored: {state_count}")
def possible_moves(state):
  b = state.index(0)
  directions = []
  if b not in [0, 1, 2]: directions.append('u')
  if b not in [6, 7, 8]: directions.append('d')
  if b not in [0, 3, 6]: directions.append('l')
  if b not in [2, 5, 8]: directions.append('r')
  return [(gen(state, d, b), d) for d in directions]
def gen(state, direction, b):
  temp = state.copy()
  if direction == 'u': temp[b], temp[b - 3] = temp[b - 3], temp[b]
  if direction == 'd': temp[b], temp[b + 3] = temp[b + 3], temp[b]
  if direction == 'I': temp[b], temp[b - 1] = temp[b - 1], temp[b]
  if direction == 'r': temp[b], temp[b + 1] = temp[b + 1], temp[b]
  return temp
```

def print\_board(state):

board = np.array(state).reshape(3, 3)
print(board)

# Initial configuration and target configuration

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

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

# Run BFS to solve the puzzle

bfs(src, target)

### **Output:**

```
[[1 2 3]
  [0 4 6]
  [7 5 8]]
 [[0 2 3]
 [1 4 6]
 [7 5 8]]
 Current move: u
 [[1 2 3]
 [7 4 6]
 [0 5 8]]
 Current move: d
 [[1 2 3]
 [4 0 6]
 [7 5 8]]
 Current move: r
 [[2 0 3]
 [1 4 6]
 [7 5 8]]
 Current move: r
 [[1 2 3]
 [7 4 6]
 [5 0 8]]
 Current move: r
 [[1 0 3]
 [4 2 6]
 [7 5 8]]
 Current move: u
 [[1 2 3]
 [4 5 6]
 [7 0 8]]
 Current move: d
 [[1 2 3]
 [4 6 0]
  [7 5 8]]
 Current move: r
```

```
[[2 4 3]
[1 0 6]
 [7 5 8]]
Current move: d
[[2 3 0]
[1 4 6]
[7 5 8]]
Current move: r
[[1 2 3]
[7 0 6]
[5 4 8]]
Current move: u
[[1 2 3]
[7 4 6]
[5 8 0]]
Current move: r
[[0 1 3]
[4 2 6]
 [7 5 8]]
Current move: 1
[[1 3 0]
[4 2 6]
[7 5 8]]
Current move: r
[[1 2 3]
[4 5 6]
[0 7 8]]
Current move: 1
[[1 2 3]
[4 5 6]
 [7 8 0]]
Current move: r
Goal state achieved!
Total unique states explored: 17
```