Week 3

Solving 8 puzzle using BFS and DFS.

CODE:

```
from collections import deque

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

def find_zero(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def get_neighbors(state):
    x, y = find_zero(state)
    moves = [(-1,0),(1,0),(0,-1),(0,1)]
    neighbors = []
    for dx, dy in moves:
        nx, ny = x+dx, y+dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny],
new_state[x][y]
            neighbors.append(new_state)
    return neighbors

def state_to_tuple(state):
    return tuple(tuple(row) for row in state)

def bfs(start):
    queue = deque([(start, [])])
    visited = set()
    while queue:
        state, path = queue.popleft()
        if state == GOAL_STATE:
            return path + [state]
        visited.add(state_to_tuple(state))
        for neighbor in get_neighbors(state):
            if state_to_tuple(neighbor) not in visited:
```

```
                queue.append((neighbor, path + [state]))
    return None

def dfs(start, max_depth=50):
    stack = [(start, [], 0)]
    visited = set()
    while stack:
        state, path, depth = stack.pop()
        if state == GOAL_STATE:
            return path + [state]
        if depth >= max_depth:
            continue
        visited.add(state_to_tuple(state))
        for neighbor in get_neighbors(state):
            if state_to_tuple(neighbor) not in visited:
                stack.append((neighbor, path + [state], depth+1))
    return None


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

print("BFS Solution:")
bfs_solution = bfs(start_state)
if bfs_solution:
    for step in bfs_solution:
        for row in step:
            print(row)
        print("---")
else:
    print("No solution found.")

print("DFS Solution:")
dfs_solution = dfs(start_state)
if dfs_solution:
    for step in dfs_solution:
        for row in step:
            print(row)
        print("---")
else:
    print("No solution found.")
```

```
BFS Solution:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
---
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
---
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
---
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
---
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
---
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
---
SUHAS BP (1BM23CS345)
```

```
DFS Solution:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
---
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]
---
[2, 8, 3]
[1, 6, 0]
[7, 5, 4]
---
[2, 8, 3]
[1, 0, 6]
[7, 5, 4]
---
[2, 8, 3]
[0, 1, 6]
[7, 5, 4]
---
[2, 8, 3]
[7, 1, 6]
[0, 5, 4]
---
[2, 8, 3]
[7, 1, 6]
[5, 0, 4]
---
[2, 8, 3]
[7, 1, 6]
[5, 4, 0]
---
```

SUHAS BP (1BM23CS345)