Lab 3 – 8 Puzzle Problem

Code BFS

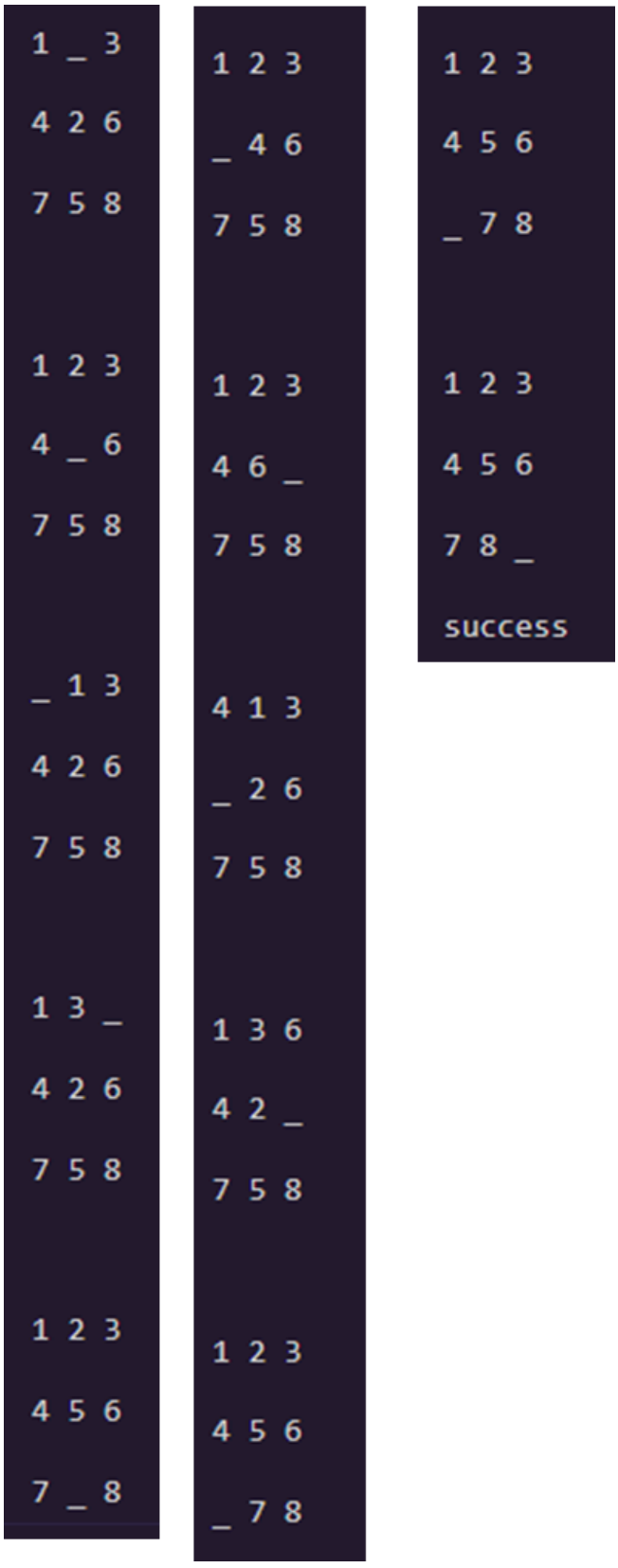
```
def bfs(src,target):

    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 BFS


1 _ 3
4 2 6
7 5 8
1 2 3
4 _ 6
7 5 8
_ 1 3
4 2 6
7 5 8
1 3 _
4 2 6
7 5 8
1 2 3
4 5 6
7 _ 8
1 2 3
_ 4 6
7 5 8
1 2 3
4 6 _
7 5 8
4 1 3
_ 2 6
7 5 8
1 3 6
4 2 _
7 5 8
1 2 3
4 5 6
_ 7 8
1 2 3
4 5 6
_ 7 8
1 2 3
4 5 6
7 8 _
success

Code DFS

```
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 DFS

```
Current state:
1 2 3
0 4 6
7 5 8

Took a U move
Current state:
0 2 3
1 4 6
7 5 8

Took a R move
Current state:
2 0 3
1 4 6
7 5 8

Took a R move
Current state:
2 3 0
1 4 6
7 5 8

Took a D move
Current state:
2 3 6
1 4 0
7 5 8

Took a D move
Current state:
2 3 6
1 4 8
7 5 0

Took a L move
Current state:
2 3 6
1 4 8
7 0 5

Took a U move
Current state:
2 3 6
1 0 8
7 4 5

Took a L move
Current state:
2 3 6
1 4 8
0 7 5
```

```
Took a L move
Current state:
2 3 6
1 0 4
7 5 8

Took a D move
Current state:
2 3 6
1 5 4
7 0 8

Took a R move
Current state:
2 3 6
1 5 4
7 8 0

Took a L move
Current state:
2 3 6
1 5 4
0 7 8

Took a D move
Current state:
2 4 3
1 0 6
7 5 8

Took a R move
Current state:
2 4 3
1 6 0
7 5 8

Took a U move
Current state:
2 4 0
1 6 3
7 5 8

Took a D move
Current state:
2 4 3
1 6 8
7 5 0

Took a L move
Current state:
2 4 3
1 6 8
7 0 5
```

```
Took a L move
Current state:
2 4 3
1 6 8
0 7 5

Took a D move
Current state:
2 4 3
1 5 6
7 0 8

Took a R move
Current state:
2 4 3
1 5 6
7 8 0

Took a U move
Current state:
2 4 3
1 5 0
7 8 6

Took a U move
Current state:
2 4 0
1 5 3
7 8 6

Took a L move
Current state:
2 4 3
1 5 6
0 7 8

Took a R move
Current state:
1 2 3
4 0 6
7 5 8

Took a U move
Current state:
1 0 3
4 2 6
7 5 8

Took a R move
Current state:
1 3 0
4 2 6
7 5 8
```

```
Took a D move
Current state:
1 3 6
4 2 8
7 5 0

Took a L move
Current state:
1 3 6
4 2 8
7 0 5

Took a L move
Current state:
1 3 6
4 2 8
0 7 5

Took a L move
Current state:
0 1 3
4 2 6
7 5 8

Took a R move
Current state:
1 2 3
4 6 0
7 5 8

Took a U move
Current state:
1 2 0
4 6 3
7 5 8

Took a L move
Current state:
1 0 2
4 6 3
7 5 8

Took a L move
Current state:
0 1 2
4 6 3
7 5 8

Took a D move
Current state:
1 2 3
4 6 8
7 5 0
```

```
Took a L move
Current state:
1 2 3
4 6 8
7 0 5

Took a L move
Current state:
1 2 3
4 6 8
0 7 5

Took a D move
Current state:
1 2 3
4 5 6
7 0 8

Took a R move
Current state:
1 2 3
4 5 6
7 8 0

1 2 3
4 5 6
7 8 0

Number of states : 42
Solution found: True
```