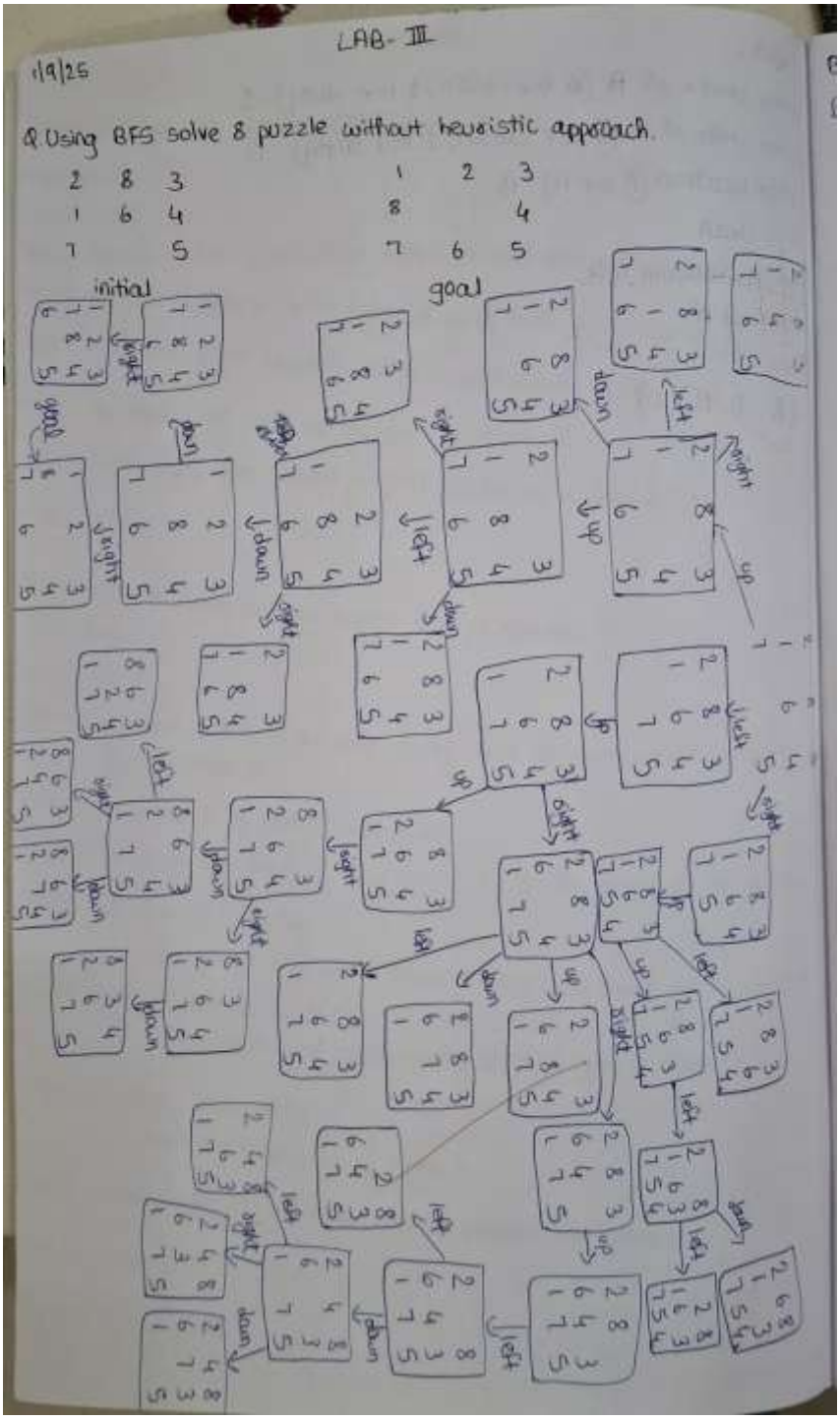
Lab 2

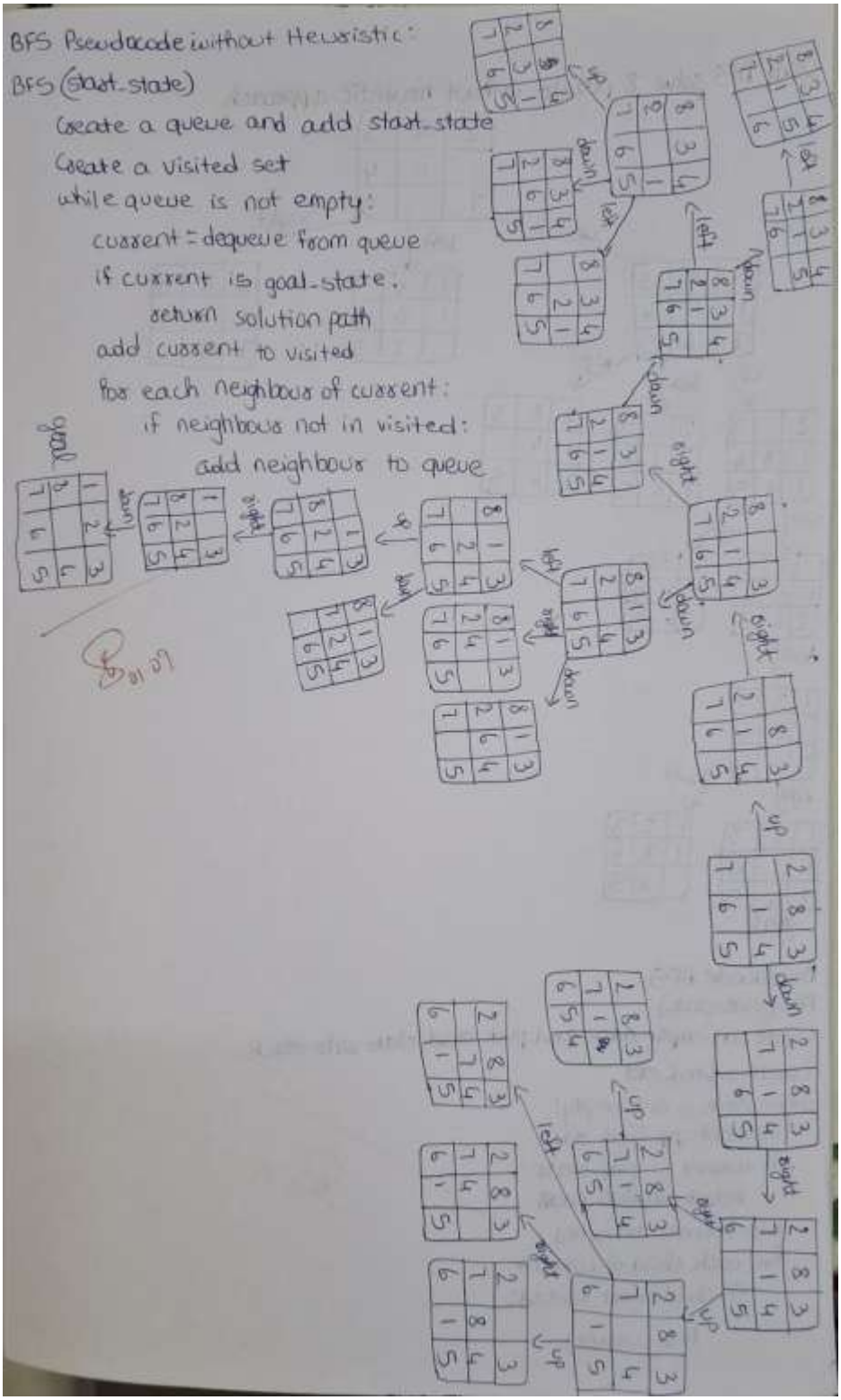
Using BFS solve 8 puzzle without heuristic approach.

Algorithm:



BFS Pseudocode without Heuristic:

```
BFS (start_state)
    Create a queue and add start_state
    Create a visited set
    while queue is not empty:
        current = dequeue from queue
        if current is goal_state:
            return solution path
        add current to visited
        for each neighbour of current:
            if neighbour not in visited:
                add neighbour to queue
```



Code:

```
print("Shreya Raj 1BM23CS317")
from collections import deque

def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print()

def bfs(start, goal):
    queue = deque([(start, [])])
    visited = set([start])
    while queue:
        state, path = queue.popleft()
        if state == goal:
            return path + [state]
        zero = state.index(0)
        moves = []
        if zero % 3 > 0:
            moves.append(zero - 1)
        if zero % 3 < 2:
            moves.append(zero + 1)
        if zero // 3 > 0:
            moves.append(zero - 3)
        if zero // 3 < 2:
            moves.append(zero + 3)
```

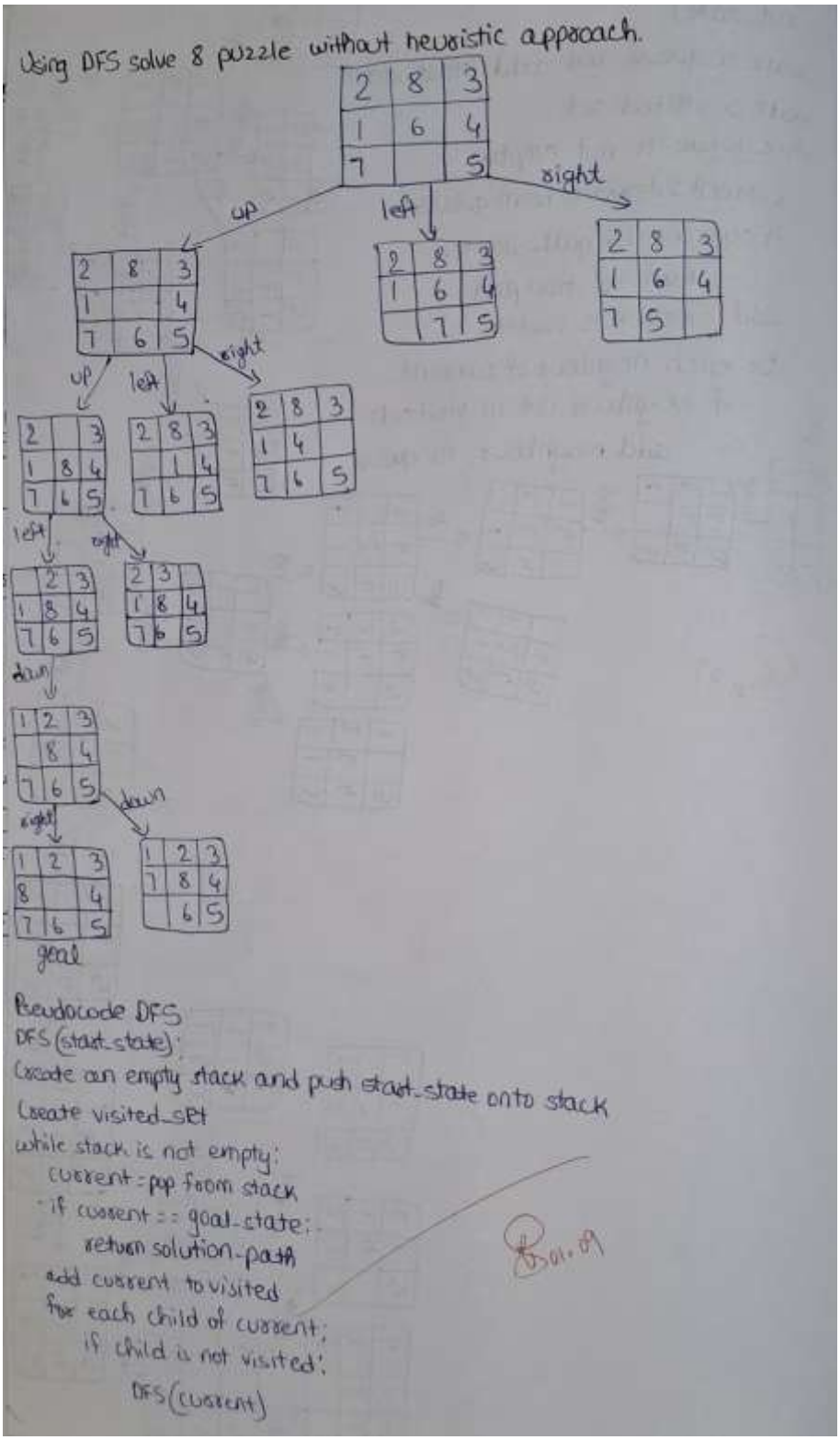
```
        for move in moves:
            new_state = list(state)
            new_state[zero], new_state[move] = new_state[move], new_state[zero]
            new_state = tuple(new_state)
            if new_state not in visited:
                visited.add(new_state)
                queue.append((new_state, path + [state]))
    return None

def input_state(prompt):
    s = input(prompt).strip().split()
    return tuple(map(int, s))

start = input_state("Enter initial state (9 numbers with 0 for blank): ")
goal = input_state("Enter goal state (9 numbers with 0 for blank): ")

result = bfs(start, goal)
if result:
    for step in result:
        print_state(step)
else:
    print("No solution found")
```

Output:

```
Shreya Raj 1BM23CS317
Enter initial state (9 numbers with 0 for blank): 2 8 3 1 6 4 7 0 5
Enter goal state (9 numbers with 0 for blank): 1 2 3 8 0 4 7 6 5
(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)
```

Using DFS solve 8 puzzle without heuristic approach.

Algorithm:



Code:

```
print("Shreya Raj 1BM23CS317")
def dfs_limited(state, goal, depth, path, visited):
    if state == goal:
        return path + [state]
    if depth == 0:
        return None

    visited.add(state)
    zero = state.index(0)
    moves = []
    # Determine valid moves
    if zero % 3 > 0:
        moves.append(zero - 1)
    if zero % 3 < 2:
        moves.append(zero + 1)
    if zero // 3 > 0:
        moves.append(zero - 3)
    if zero // 3 < 2:
        moves.append(zero + 3)

    for move in moves:
        new_state = list(state)
        new_state[zero], new_state[move] = new_state[move], new_state[zero]
        new_state = tuple(new_state)

        if new_state not in visited:
```

```
            result = dfs_limited(new_state, goal, depth - 1, path + [state], visited)
            if result:
                return result

    visited.remove(state)
    return None

def input_state(prompt):
    s = input(prompt).strip().split()
    return tuple(map(int, s))

def print_state(state):
    for i in range(0, 9, 3):
        print(state[i], state[i + 1], state[i + 2])
    print()
# Main Execution
start = input_state("Enter initial state (9 numbers with 0 for blank): ")
goal = input_state("Enter goal state (9 numbers with 0 for blank): ")

max_depth = 10
result = dfs_limited(start, goal, max_depth, [], set())

if result:
    print("\nSolution path:")
    for step in result:
        print_state(step)
else:
```

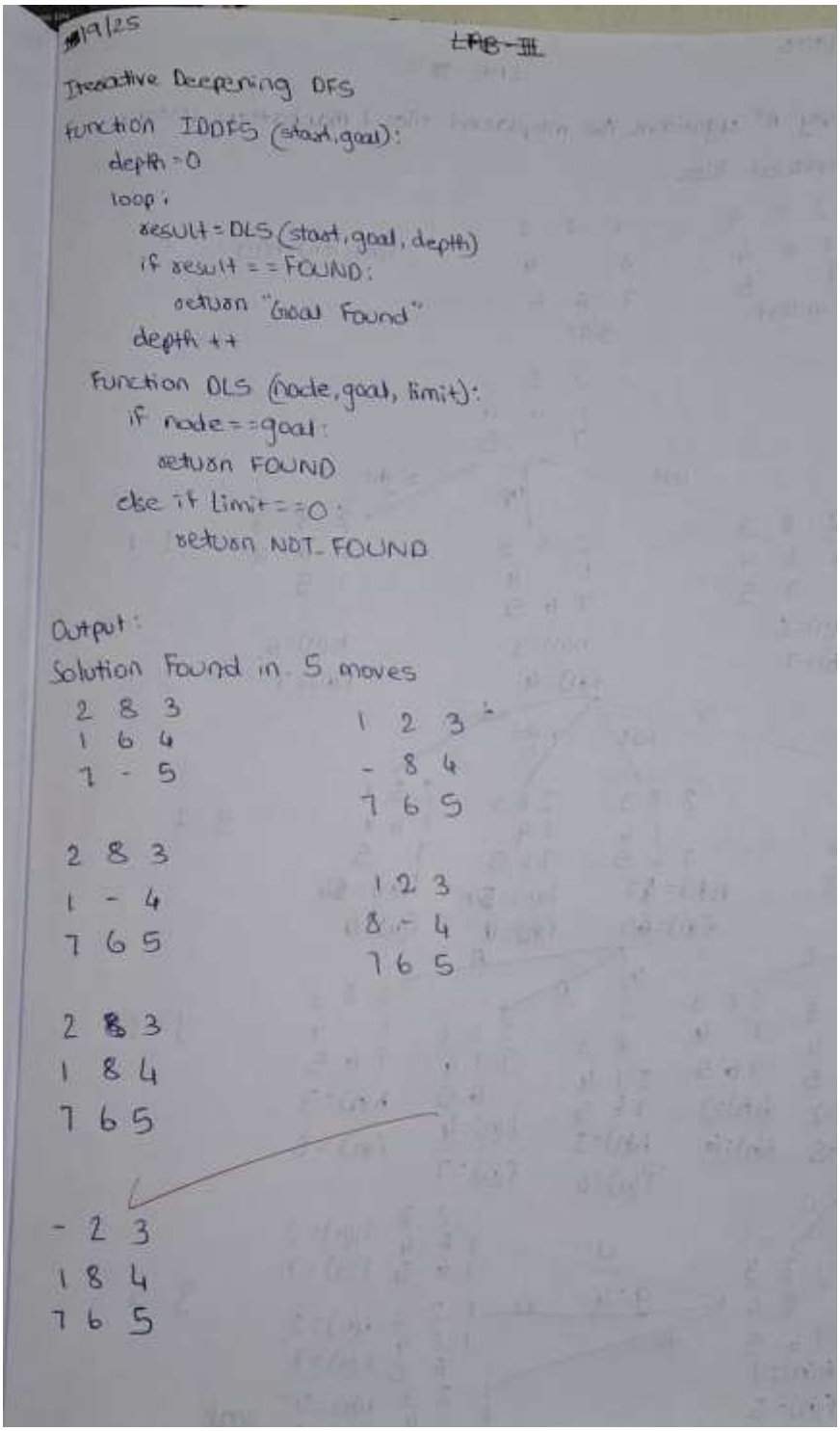
```
    print("No solution found within depth limit")
```

Output:

```
Shreya Raj 1BM23CS317
Enter initial state (9 numbers with 0 for blank): 2 8 3 1 6 4 7 0 5
Enter goal state (9 numbers with 0 for blank): 1 2 3 8 0 4 7 6 5

Solution path:
2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 8 3
0 1 4
7 6 5

0 8 3
2 1 4
7 6 5

8 0 3
2 1 4
7 6 5

8 1 3
2 0 4
7 6 5

8 1 3
0 2 4
7 6 5

0 1 3
8 2 4
7 6 5

1 0 3
8 2 4
7 6 5

1 2 3
8 0 4
7 6 5
```

Using Iterative Deepening DFS solve 8 puzzle without heuristic approach.

Algorithm:



19/25

~~LAB-III~~

Iterative Deepening DFS

```
function IDDFS (start, goal):
    depth = 0
    loop:
      result = DLS (start, goal, depth)
      if result == FOUND:
        return "Goal Found"
      depth ++
  Function DLS (node, goal, limit):
    if node == goal:
      return FOUND
    else if limit == 0:
      return NOT_FOUND
```

Output:

Solution Found in 5 moves

```
2 8 3        1 2 3
1 6 4        - 8 4
7 - 5        7 6 5

2 8 3        1 2 3
1 - 4        8 - 4
7 6 5        7 6 5

2 - 3
1 8 4
7 6 5

- 2 3
1 8 4
7 6 5
```

Code:

```
print("Shreya Raj 1BM23CS317")
class PuzzleState:
    def __init__(self, board, empty_pos, moves=0, path=None):
        self.board = board
        self.empty_pos = empty_pos
        self.moves = moves
        self.path = path or [board]

    def is_goal(self, goal):
        return self.board == goal

    def get_neighbors(self):
        neighbors = []
        x, y = self.empty_pos
        directions = [(-1,0),(1,0),(0,-1),(0,1)]  # Up, Down, Left, Right
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 3 and 0 <= ny < 3:
                new_board = [list(row) for row in self.board]
                # swap empty_pos with target
                new_board[x][y], new_board[nx][ny] = new_board[nx][ny], new_board[x][y]
                new_board = tuple(tuple(row) for row in new_board)
                neighbors.append(PuzzleState(new_board, (nx, ny), self.moves + 1, self.path +
[new_board]))
        return neighbors
```

```
def dls(state, goal, limit, visited):
    if state.is_goal(goal):
        return state.path
    if limit == 0:
        return None
    visited.add(state.board)
    for neighbor in state.get_neighbors():
        if neighbor.board not in visited:
            result = dls(neighbor, goal, limit - 1, visited)
            if result is not None:
                return result
    visited.remove(state.board)
    return None

def iddfs(start, goal):
    depth = 0
    while True:
        visited = set()
        result = dls(start, goal, depth, visited)
        if result is not None:
            return result
        depth += 1

def print_path(path):
    print(f"Solution Found in {len(path)-1} moves")
    for state in path:
        for row in state:
```

```
            print(" ".join(str(x) if x != 0 else "-" for x in row))
        print()

def get_user_board(prompt):
    print(prompt)
    board = []
    for i in range(3):
        row = list(map(int, input(f"Row {i+1} (space separated, use 0 for empty): ").strip().split()))
        board.append(tuple(row))
    return tuple(board)

start_board = get_user_board("Enter the initial state:")
goal_board = get_user_board("Enter the goal state:")

# Locate empty in start state
empty_pos = None
for i in range(3):
    for j in range(3):
        if start_board[i][j] == 0:
            empty_pos = (i, j)
            break
    if empty_pos is not None:
        break

start_state = PuzzleState(start_board, empty_pos)
path = iddfs(start_state, goal_board)
if path:
```

```
    print_path(path)
else:
    print("No solution found.")
```

Output:

```
Shreya Raj 1BM23CS317
Enter the initial state:
Row 1 (space separated, use 0 for empty): 2 8 3
Row 2 (space separated, use 0 for empty): 1 6 4
Row 3 (space separated, use 0 for empty): 7 0 5
Enter the goal state:
Row 1 (space separated, use 0 for empty): 1 2 3
Row 2 (space separated, use 0 for empty): 8 0 4
Row 3 (space separated, use 0 for empty): 7 6 5
Solution Found in 5 moves
2 8 3
1 6 4
7 - 5

2 8 3
1 - 4
7 6 5

2 - 3
1 8 4
7 6 5

- 2 3
1 8 4
7 6 5

1 2 3
- 8 4
7 6 5

1 2 3
8 - 4
7 6 5
```