BFS 8 PUZZLE SOLVE

CODE :

```
print(f"Santhosh N (1BM23CS302)")
from collections import deque

def print_state(state):
    for row in state:
        print(' '.join(str(x) for x in row))
    print()

def is_goal(state, goal_state):
    return state == goal_state

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):
    neighbors = []
    x, y = find_zero(state)
    directions = [(1,0), (-1,0), (0,1), (0,-1)]
    for dx, dy in directions:
        new_x, new_y = x + dx, y + dy
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y]
            neighbors.append(new_state)
    return neighbors

def bfs(start_state, goal_state):
    queue = deque()
    queue.append((start_state, [start_state]))
    visited = set()
    visited.add(tuple(tuple(row) for row in start_state))

    visited_count = 0

    print("Starting BFS traversal...\n")

    while queue:
        current_state, path = queue.popleft()
```

BFS 8 PUZZLE SOLVE

```
        visited_count += 1
        print(f"Visited state #{visited_count}:")
        print_state(current_state)

        if is_goal(current_state, goal_state):
            print(f"Total visited states: {visited_count}")
            return path

        for neighbor in get_neighbors(current_state):
            neighbor_tuple = tuple(tuple(row) for row in neighbor)
            if neighbor_tuple not in visited:
                visited.add(neighbor_tuple)
                queue.append((neighbor, path + [neighbor]))

    print(f"Total visited states: {visited_count}")
    return None

def read_state(name):
    print(f"Enter the {name} state, row by row (use space-separated numbers, 0 for empty):")
    state = []
    for _ in range(3):
        row = input().strip().split()
        if len(row) != 3:
            raise ValueError("Each row must have exactly 3 numbers.")
        row = list(map(int, row))
        state.append(row)
    return state


initial_state = read_state("initial")
goal_state = read_state("goal")

solution_path = bfs(initial_state, goal_state)

if solution_path:
    cost = len(solution_path) - 1
    print(f"\nSolution found with cost: {cost}\n")
    print("Solution path:")
    for state in solution_path:
        print_state(state)
else:
    print("No solution found")
```

BFS 8 PUZZLE SOLVE

OUTPUT :

```
= RESTART: C:/Users/student/AppData/Local/Programs/Python/Python313/302/lab3_bfs.py
Santhosh N (1BM23CS302)
Enter the initial state, row by row (use space-separated numbers, 0 for empty):
2 8 3
1 6 4
7 0 5
Enter the goal state, row by row (use space-separated numbers, 0 for empty):
1 2 3
8 0 4
7 6 5
Starting BFS traversal...

Visited state #1:
2 8 3
1 6 4
7 0 5

Visited state #2:
2 8 3
1 0 4
7 6 5

Visited state #3:
2 8 3
1 6 4
7 5 0

Visited state #4:
2 8 3
1 6 4
0 7 5

Visited state #5:
2 0 3
1 8 4
7 6 5

Visited state #6:
2 8 3
1 4 0
7 6 5

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

BFS 8 PUZZLE SOLVE

```
Visited state #32:
2 0 3
6 8 4
1 7 5

Visited state #33:
2 8 3
6 4 0
1 7 5

Visited state #34:
2 3 4
1 8 5
7 6 0

Visited state #35:
2 3 4
1 0 8
7 6 5

Visited state #36:
1 2 3
7 8 4
0 6 5

Visited state #37:
1 2 3
8 0 4
7 6 5

Total visited states: 37

Solution found with cost: 5

Solution path:
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
```