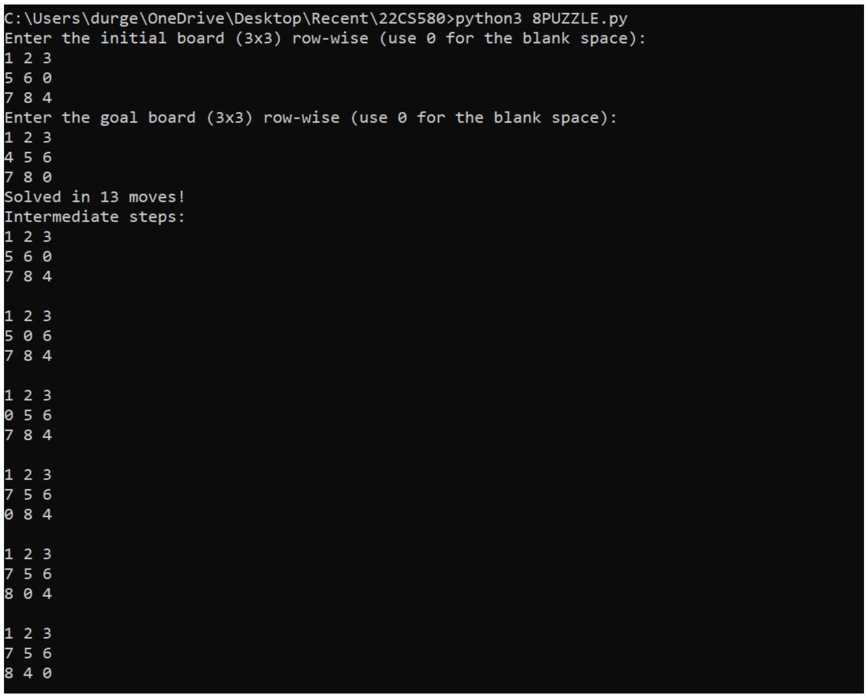
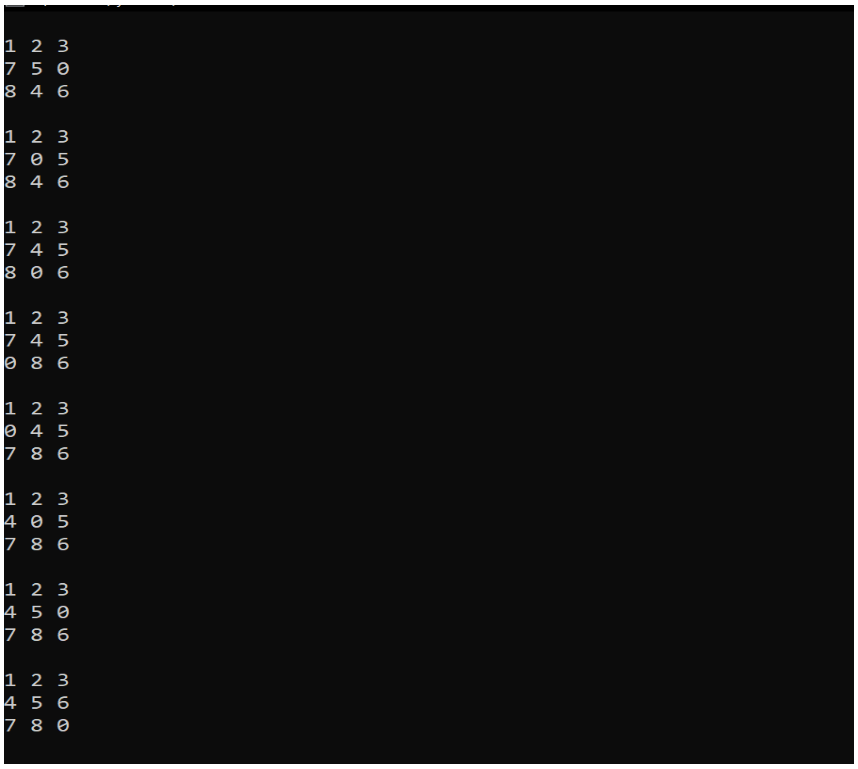
**PROGRAM:**

```
import heapq
def get_manhattan_distance(board, goal):
    distance = 0
    for i in range(3):
        for j in range(3):
            if board[i][j] != 0:
                x, y = divmod(goal.index(board[i][j]), 3)
                distance += abs(x - i) + abs(y - j)
    return distance
def get_neighbors(board):
    neighbors = []
    x, y = [(i, j) for i in range(3) for j in range(3) if board[i][j] == 0][0]
    moves = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
    for i, j in moves:
        if 0 <= i < 3 and 0 <= j < 3:
            new_board = [row[:] for row in board]
            new_board[x][y], new_board[i][j] = new_board[i][j], new_board[x][y]
            neighbors.append(new_board)
    return neighbors
def best_first_search(initial, goal):
    initial_flat = [num for row in initial for num in row]
    goal_flat = [num for row in goal for num in row]
    heap = [(get_manhattan_distance(initial, goal_flat), 0, initial, [initial])]
    visited = set()
    visited.add(tuple(tuple(row) for row in initial))
     while heap:
        _, moves, current, path = heapq.heappop(heap)
        if current == goal:
            return path
        for neighbor in get_neighbors(current):
            neighbor_tuple = tuple(tuple(row) for row in neighbor)
            if neighbor_tuple not in visited:
                visited.add(neighbor_tuple)
                heapq.heappush(heap, (get_manhattan_distance(neighbor, goal_flat) + moves + 1, moves + 1,
neighbor, path + [neighbor]))
    return []
def get_input_board(prompt):
    print(prompt)
    board = []
    for _ in range(3):
        row = list(map(int, input().strip().split()))
        board.append(row)
    return board
def print_board(board):
    for row in board:
        print(" ".join(map(str, row)))
    print()
initial_board = get_input_board("Enter the initial board (3x3) row-wise (use 0 for the blank space):")
goal_board = get_input_board("Enter the goal board (3x3) row-wise (use 0 for the blank space):")
path = best_first_search(initial_board, goal_board)
if path:
    print(f"Solved in {len(path) - 1} moves!")
    print("Intermediate steps:")
    for step in path:
        print_board(step)
else:
    print("No solution found.")
```

**OUTPUT:**



```
C:\Users\durge\OneDrive\Desktop\Recent\22CS580>python3 8PUZZLE.py
Enter the initial board (3x3) row-wise (use 0 for the blank space):
1 2 3
5 6 0
7 8 4
Enter the goal board (3x3) row-wise (use 0 for the blank space):
1 2 3
4 5 6
7 8 0
Solved in 13 moves!
Intermediate steps:
1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
0 5 6
7 8 4

1 2 3
7 5 6
0 8 4

1 2 3
7 5 6
8 0 4

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



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

1 2 3
7 0 5
8 4 6

1 2 3
7 4 5
8 0 6

1 2 3
7 4 5
0 8 6

1 2 3
0 4 5
7 8 6

1 2 3
4 0 5
7 8 6

1 2 3
4 5 0
7 8 6

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