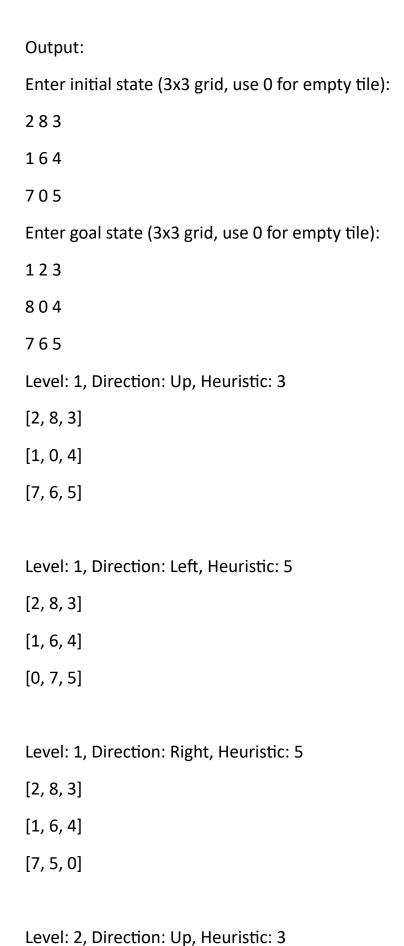
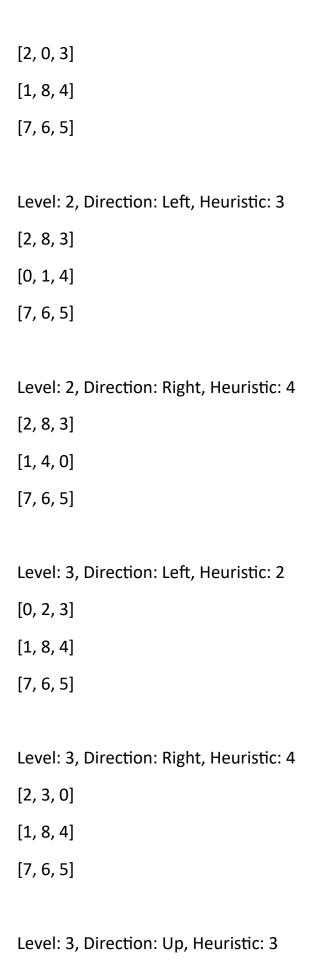
```
Python code for 8-puzzle A* implementation, to calculate, f(n),
considering:
g(n): Depth of the node, h(n): Number of Misplaced tiles
import heapq
class PuzzleState:
  def __init__(self, state, empty_tile_pos, g, h, path, level):
    self.state = state
    self.empty_tile_pos = empty_tile_pos # (row, col)
    self.g = g # Cost from start to current state
    self.h = h # Heuristic cost to goal
    self.f = g + h \# Total cost
    self.path = path # Path taken to reach this state
    self.level = level # Depth level in the state space
  def It (self, other):
    return self.f < other.f # Priority queue comparison based on f value
def astar_misplaced_tiles(start_state, goal_state):
  directions = {
    (-1, 0): 'Up',
    (1, 0): 'Down',
    (0, -1): 'Left',
    (0, 1): 'Right'
  }
```

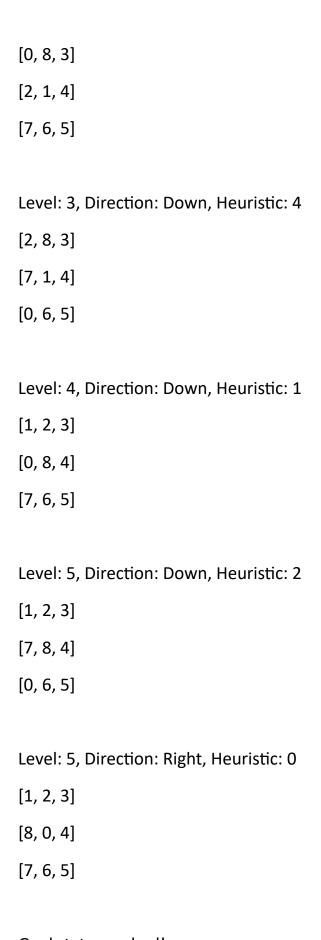
```
def calculate heuristic(state):
    h = 0
    for i in range(3):
       for j in range(3):
         if state[i][j] != goal_state[i][j] and state[i][j] != 0:
           h += 1
     return h
  def generate_moves(state, empty_tile_pos):
    moves = []
    row, col = empty_tile_pos
    for (dr, dc), direction in directions.items():
       new row, new col = row + dr, col + dc
       if 0 \le \text{new row} \le 3 and 0 \le \text{new col} \le 3:
         new state = [list(r) for r in state] # Deep copy
         # Swap the empty tile with the adjacent tile
         new state[row][col], new state[new row][new col] =
new_state[new_row][new_col], new_state[row][col]
         moves.append((new_state, (new_row, new_col), direction))
    return moves
  visited = set()
  start_empty_pos = next((i, j) for i in range(3) for j in range(3) if
start_state[i][j] == 0)
  start_h = calculate_heuristic(start_state)
```

```
start node = PuzzleState(start state, start empty pos, 0, start h,
[start_state], 0)
  priority_queue = []
  heapq.heappush(priority_queue, start_node)
  while priority_queue:
    current node = heapq.heappop(priority queue)
    # Check if we reached the goal
    if current_node.state == goal_state:
      print("Goal state reached!")
      for step in current_node.path:
        for row in step:
           print(row)
         print()
      return
    visited.add(tuple(map(tuple, current_node.state))) # Add current state to
visited
    # Generate possible moves
    for new_state, new_empty_pos, direction in
generate_moves(current_node.state, current_node.empty_tile_pos):
      if tuple(map(tuple, new_state)) not in visited:
        g = current_node.g + 1 # Cost from start
```

```
h = calculate heuristic(new state) # Heuristic
         new path = current node.path + [new state]
         new node = PuzzleState(new state, new empty pos, g, h, new path,
current_node.level + 1)
        # Print state information
         print(f"Level: {new_node.level}, Direction: {direction}, Heuristic:
{new node.h}")
        for row in new_node.state:
           print(row)
         print()
         heapq.heappush(priority queue, new node)
def main():
  print("Enter initial state (3x3 grid, use 0 for empty tile):")
  start state = [list(map(int, input().split())) for in range(3)]
  print("Enter goal state (3x3 grid, use 0 for empty tile):")
  goal_state = [list(map(int, input().split())) for _ in range(3)]
  astar_misplaced_tiles(start_state, goal_state)
if __name__ == "__main__":
  main()
```







Goal state reached!

- [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]