```
def misplace(src, target):
    queue = []
    queue.append((src, 0))  # Store the state and its level (depth)
    visited = []

    while len(queue) > 0:
        source, level = queue.pop(0)  # First possible combination with level (g(n))
        visited.append(source)

        # Calculate misplaced tiles (h(n))
        misplaced_cost = count_misplaced(source, target)

        # Calculate the total cost f(n) = g(n) + h(n)
        total_cost = level + misplaced_cost

        print("Current State:")
        print_matrix(source)
        print(f"Level (g(n)): {level}, Misplaced Tiles (h(n)): {misplaced_cost}, Total Cost (f(n)): {total_cost}")

        if source == target:
            print("Successful")
            return

        # Get the possible moves (new states)
        possible_moves_to_do = possible_moves(source, visited, target)

        for move in possible_moves_to_do:
            if move not in visited and move not in queue:
                queue.append((move, level + 1))  # Increment level for the next state

def possible_moves(present, visited, target):
    b = present.index(0)  # Index of empty spot
    d = []  # Directions

    # Add valid directions based on the empty spot
    if b not in [0, 1, 2]:  # Can move up
        d.append('u')
    if b not in [6, 7, 8]:  # Can move down
        d.append('d')
    if b not in [0, 3, 6]:  # Can move left
        d.append('l')
    if b not in [2, 5, 8]:  # Can move right
        d.append('r')

    possible_moves_list = []
    for direction in d:
        possible_moves_list.append(move(present, direction, b))

    # Now we will choose the move that minimizes misplaced tiles
    min_misplaced = float('inf')
    best_moves = []

    for new_state in possible_moves_list:
        misplaced_count = count_misplaced(new_state, target)
        if misplaced_count < min_misplaced:
            min_misplaced = misplaced_count
            best_moves = [new_state]
        elif misplaced_count == min_misplaced:
            best_moves.append(new_state)

    # Return the best moves that haven't been visited
    return [move_it for move_it in best_moves if move_it not in visited]

def count_misplaced(state, target):
    """Count the number of misplaced tiles compared to the target."""
    return sum(1 for i in range(len(state)) if state[i] != target[i] and state[i] != 0)

def move(present, i, b):
    """Move the empty space in the given direction."""
    temp = present.copy()

    # Fixing the variable 'm' to be the direction 'i'
    if i == 'd':  # Move down
        temp[b + 3], temp[b] = temp[b], temp[b + 3]
    elif i == 'u':  # Move up
        temp[b - 3], temp[b] = temp[b], temp[b - 3]
    elif i == 'l':  # Move left
        temp[b - 1], temp[b] = temp[b], temp[b - 1]
    elif i == 'r':  # Move right
        temp[b + 1], temp[b] = temp[b], temp[b + 1]

    return temp

def convert_to_matrix(state):
    """Convert a flat list state into a 3x3 matrix."""
    return [state[i:i + 3] for i in range(0, 9, 3)]

def print_matrix(state):
    """Print the matrix representation of the state."""
    matrix = convert_to_matrix(state)
    for row in matrix:
        print(row)
    print()

# Example usage:
src = [2, 8, 3, 1, 6, 4, 7, 0, 5]
target = [1, 2, 3, 8, 0, 4, 7, 6, 5]
misplace(src, target)
```

```
Current State:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]

Level (g(n)): 0, Misplaced Tiles (h(n)): 4, Total Cost (f(n)): 4
Current State:
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]

Level (g(n)): 1, Misplaced Tiles (h(n)): 3, Total Cost (f(n)): 4
Current State:
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

Level (g(n)): 2, Misplaced Tiles (h(n)): 3, Total Cost (f(n)): 5
Current State:
[2, 8, 3]
[0, 1, 4]
[7, 6, 5]

Level (g(n)): 2, Misplaced Tiles (h(n)): 3, Total Cost (f(n)): 5
Current State:
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

Level (g(n)): 3, Misplaced Tiles (h(n)): 2, Total Cost (f(n)): 5
Current State:
[0, 8, 3]
[2, 1, 4]
[7, 6, 5]

Level (g(n)): 3, Misplaced Tiles (h(n)): 3, Total Cost (f(n)): 6
Current State:
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

Level (g(n)): 4, Misplaced Tiles (h(n)): 1, Total Cost (f(n)): 5
Current State:
[8, 0, 3]
[2, 1, 4]
[7, 6, 5]

Level (g(n)): 4, Misplaced Tiles (h(n)): 3, Total Cost (f(n)): 7
Current State:
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]

Level (g(n)): 5, Misplaced Tiles (h(n)): 0, Total Cost (f(n)): 5
Successful
```