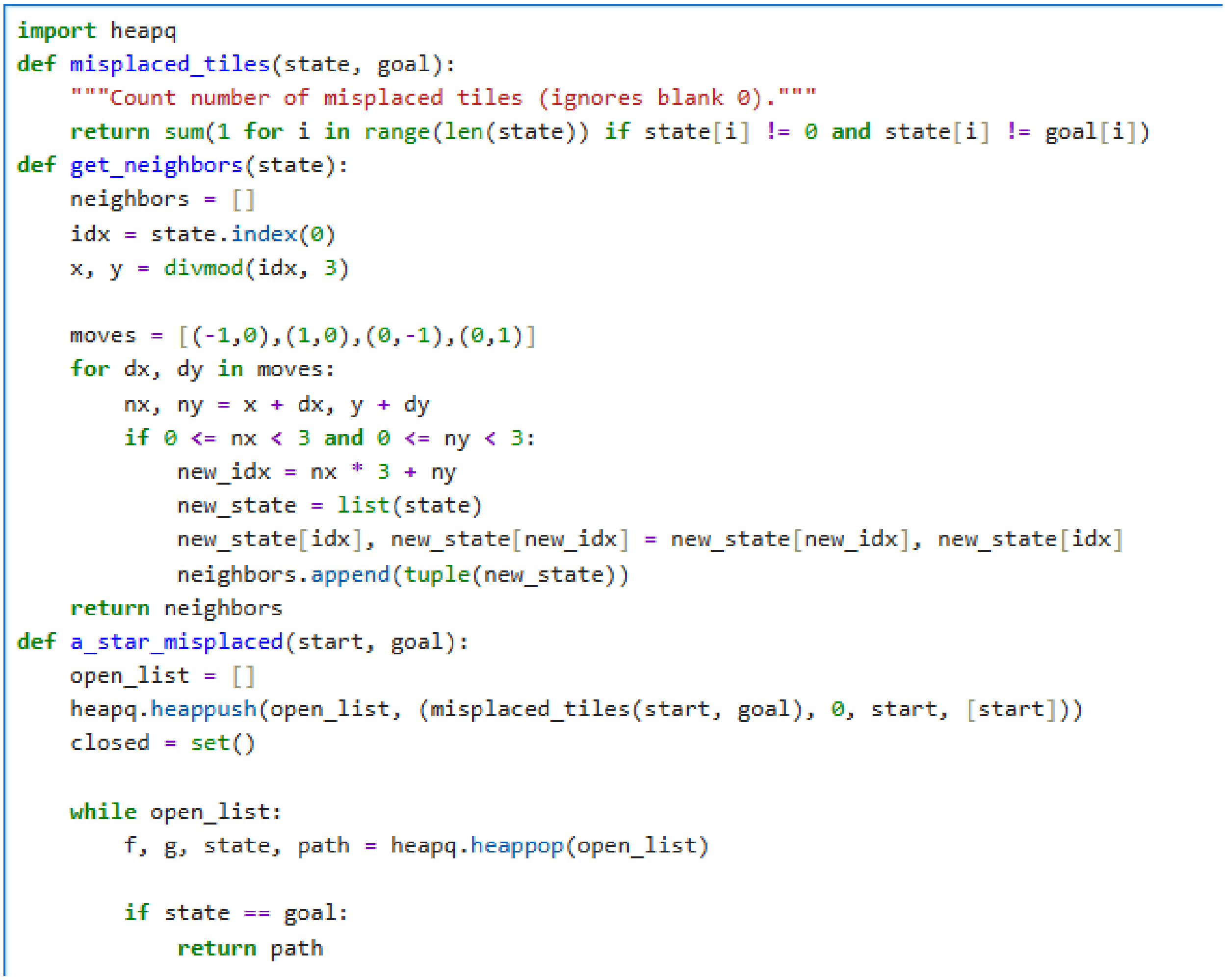
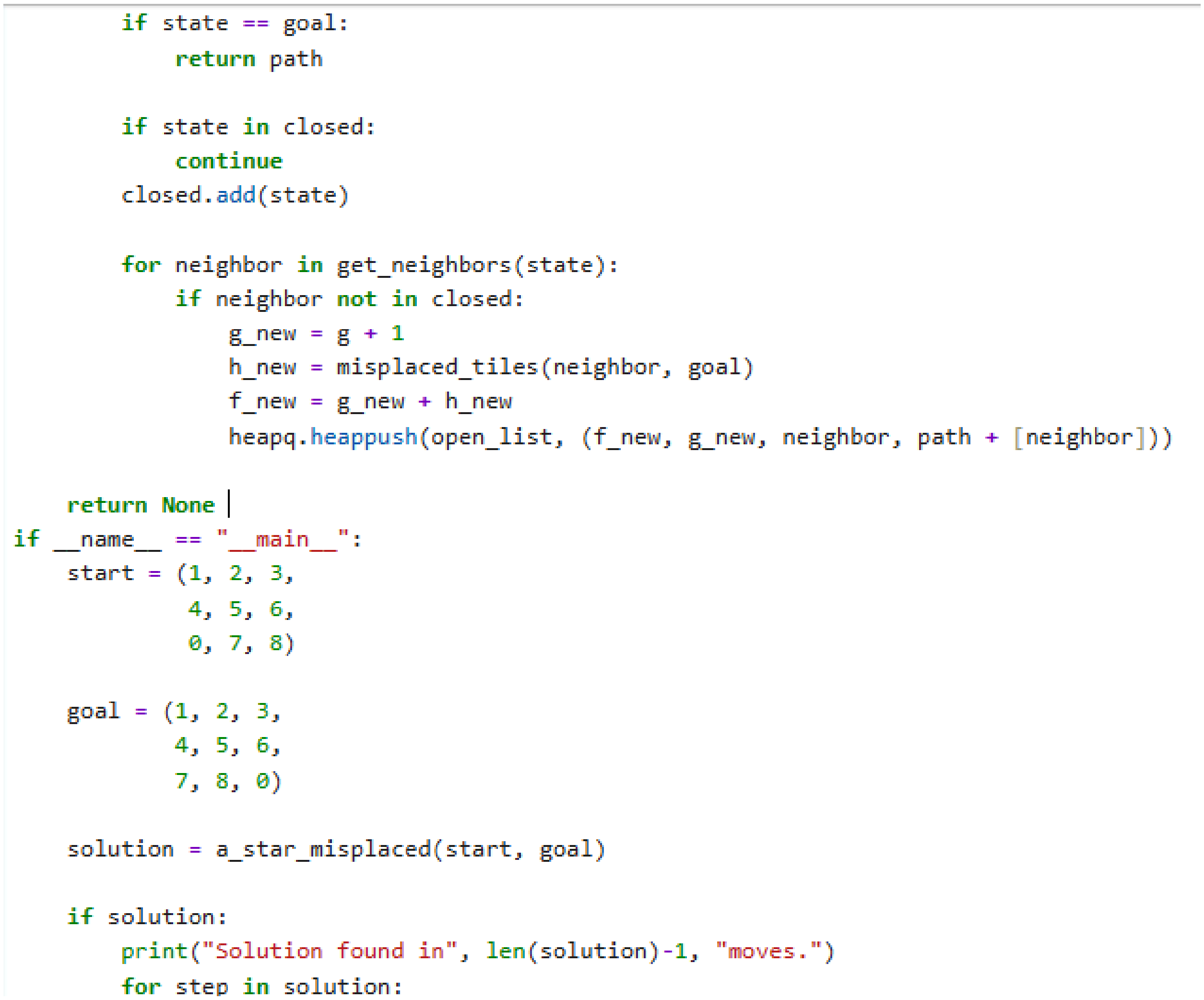
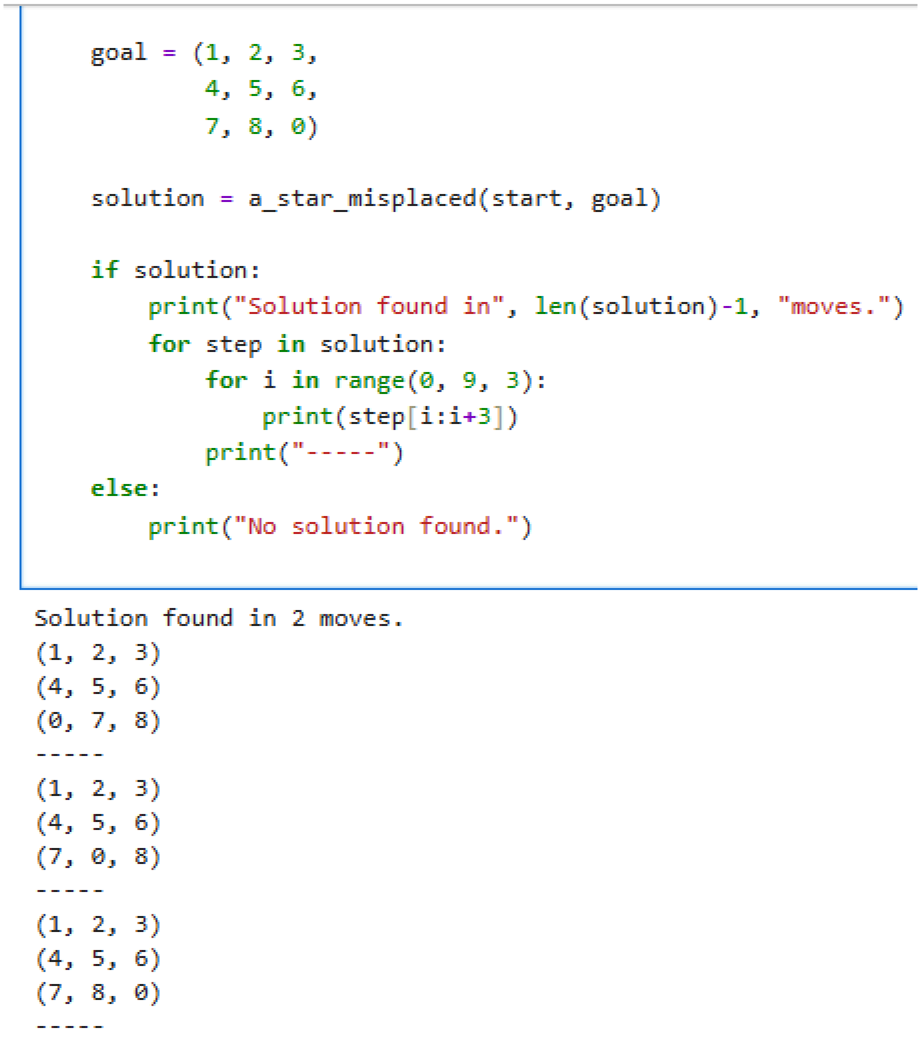
LAB 2
8 puzzle misplaced tiles



```
import heapq
def misplaced_tiles(state, goal):
    """Count number of misplaced tiles (ignores blank 0)."""
    return sum(1 for i in range(len(state)) if state[i] != 0 and state[i] != goal[i])
def get_neighbors(state):
    neighbors = []
    idx = state.index(0)
    x, y = divmod(idx, 3)

    moves = [(-1,0),(1,0),(0,-1),(0,1)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_idx = nx * 3 + ny
            new_state = list(state)
            new_state[idx], new_state[new_idx] = new_state[new_idx], new_state[idx]
            neighbors.append(tuple(new_state))
    return neighbors
def a_star_misplaced(start, goal):
    open_list = []
    heapq.heappush(open_list, (misplaced_tiles(start, goal), 0, start, [start]))
    closed = set()

    while open_list:
        f, g, state, path = heapq.heappop(open_list)

        if state == goal:
            return path
```



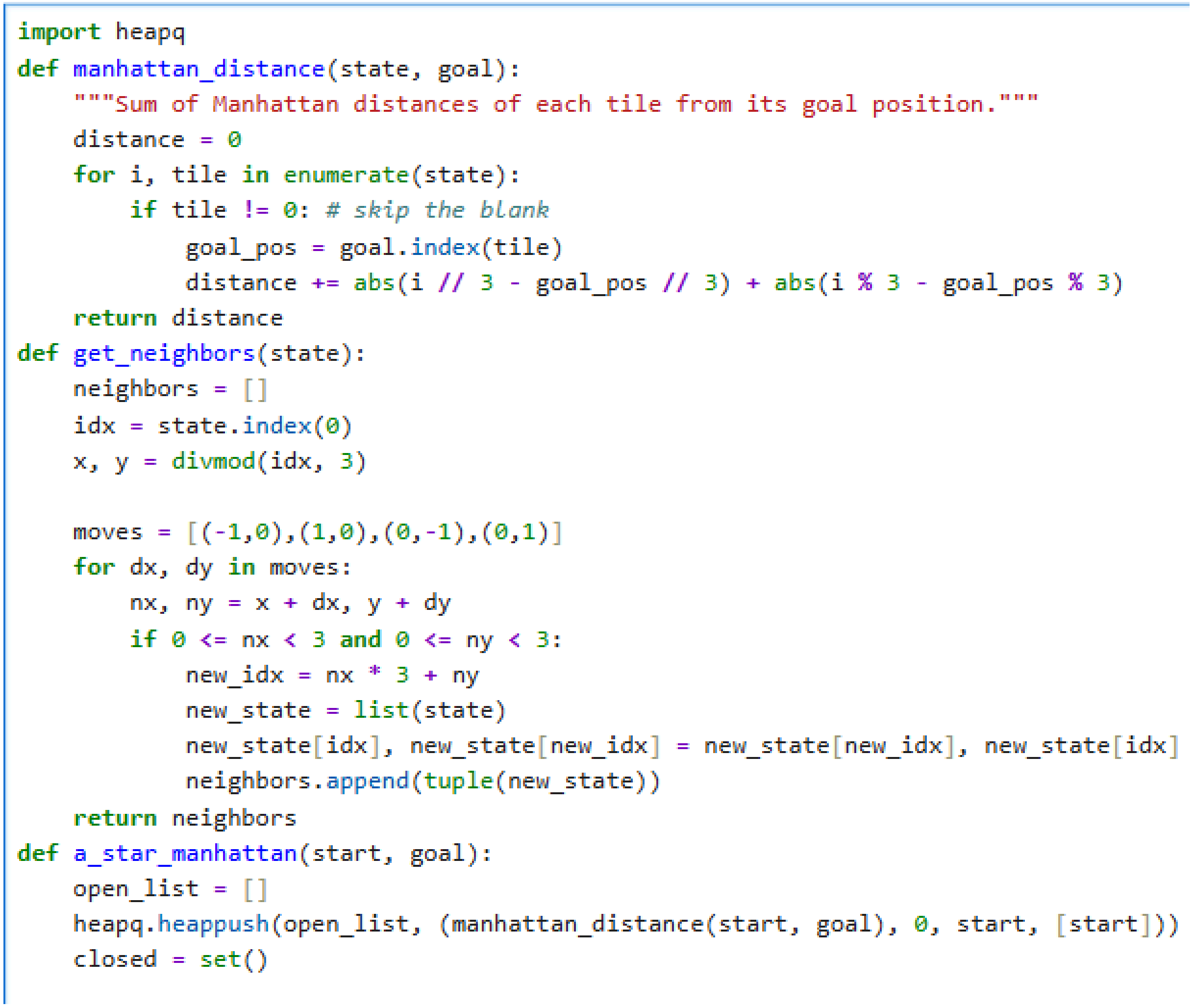
```
        if state == goal:
            return path

        if state in closed:
            continue
        closed.add(state)

        for neighbor in get_neighbors(state):
            if neighbor not in closed:
                g_new = g + 1
                h_new = misplaced_tiles(neighbor, goal)
                f_new = g_new + h_new
                heapq.heappush(open_list, (f_new, g_new, neighbor, path + [neighbor]))

    return None
if __name__ == "__main__":
    start = (1, 2, 3,
             4, 5, 6,
             0, 7, 8)

    goal = (1, 2, 3,
            4, 5, 6,
            7, 8, 0)

    solution = a_star_misplaced(start, goal)

    if solution:
        print("Solution found in", len(solution)-1, "moves.")
        for step in solution:
```



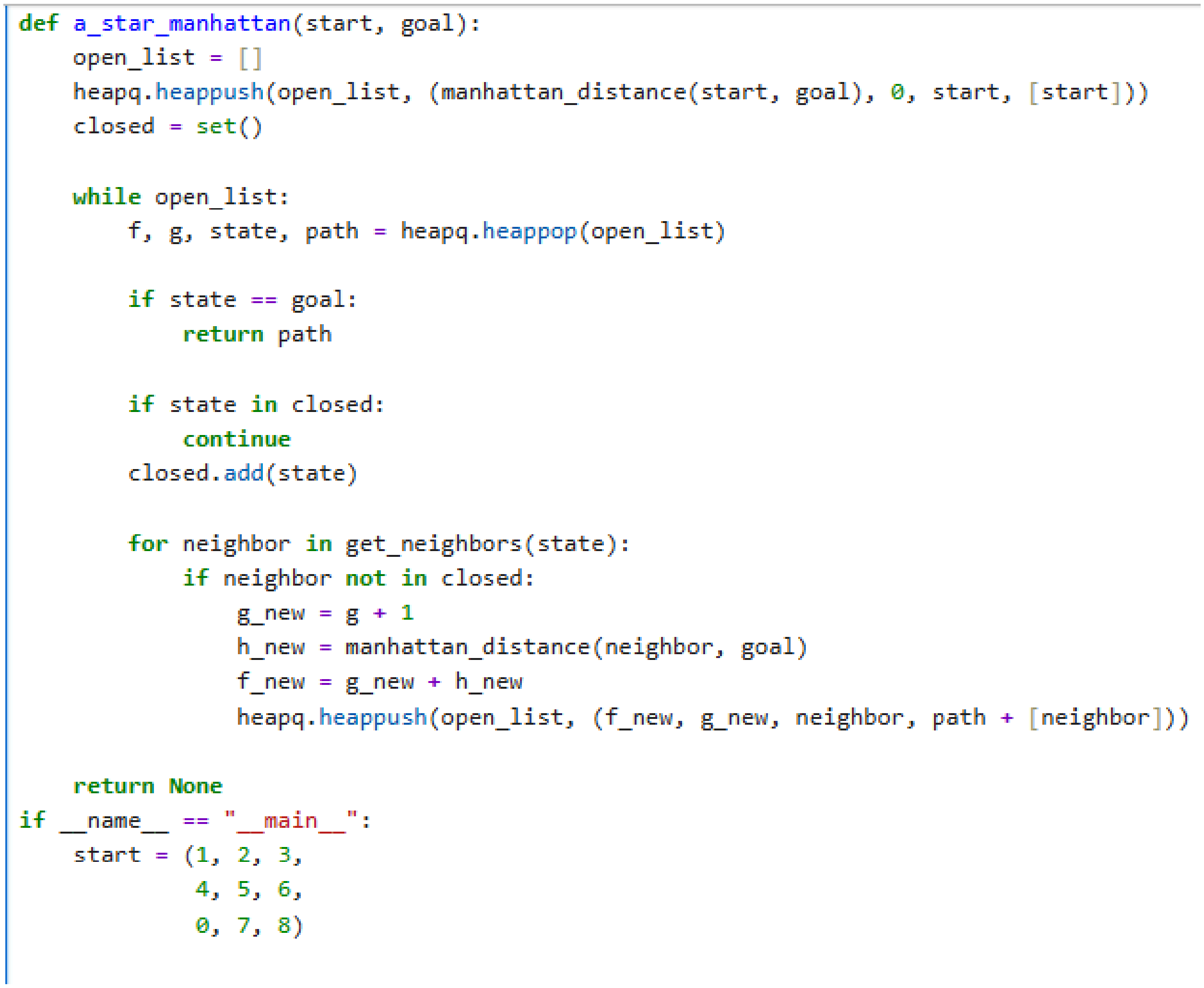
```python
goal = (1, 2, 3,
        4, 5, 6,
        7, 8, 0)

solution = a_star_misplaced(start, goal)

if solution:
    print("Solution found in", len(solution)-1, "moves.")
    for step in solution:
        for i in range(0, 9, 3):
            print(step[i:i+3])
        print("-----")
else:
    print("No solution found.")
```

```
Solution found in 2 moves.
(1, 2, 3)
(4, 5, 6)
(0, 7, 8)
-----
(1, 2, 3)
(4, 5, 6)
(7, 0, 8)
-----
(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
-----
```

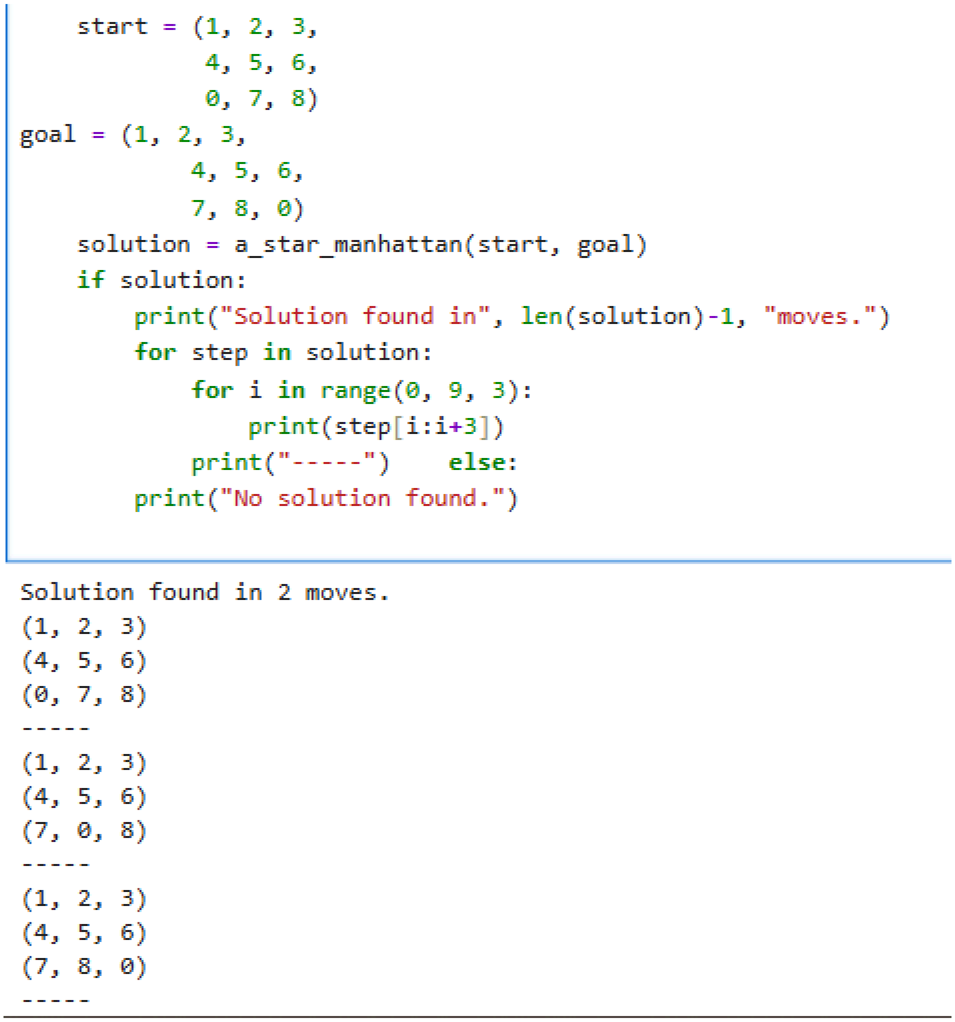
8 puzzle Manhattan distance



```python
import heapq
def manhattan_distance(state, goal):
    """Sum of Manhattan distances of each tile from its goal position."""
    distance = 0
    for i, tile in enumerate(state):
        if tile != 0: # skip the blank
            goal_pos = goal.index(tile)
            distance += abs(i // 3 - goal_pos // 3) + abs(i % 3 - goal_pos % 3)
    return distance
def get_neighbors(state):
    neighbors = []
    idx = state.index(0)
    x, y = divmod(idx, 3)

    moves = [(-1,0),(1,0),(0,-1),(0,1)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_idx = nx * 3 + ny
            new_state = list(state)
            new_state[idx], new_state[new_idx] = new_state[new_idx], new_state[idx]
            neighbors.append(tuple(new_state))
    return neighbors
def a_star_manhattan(start, goal):
    open_list = []
    heapq.heappush(open_list, (manhattan_distance(start, goal), 0, start, [start]))
    closed = set()
```



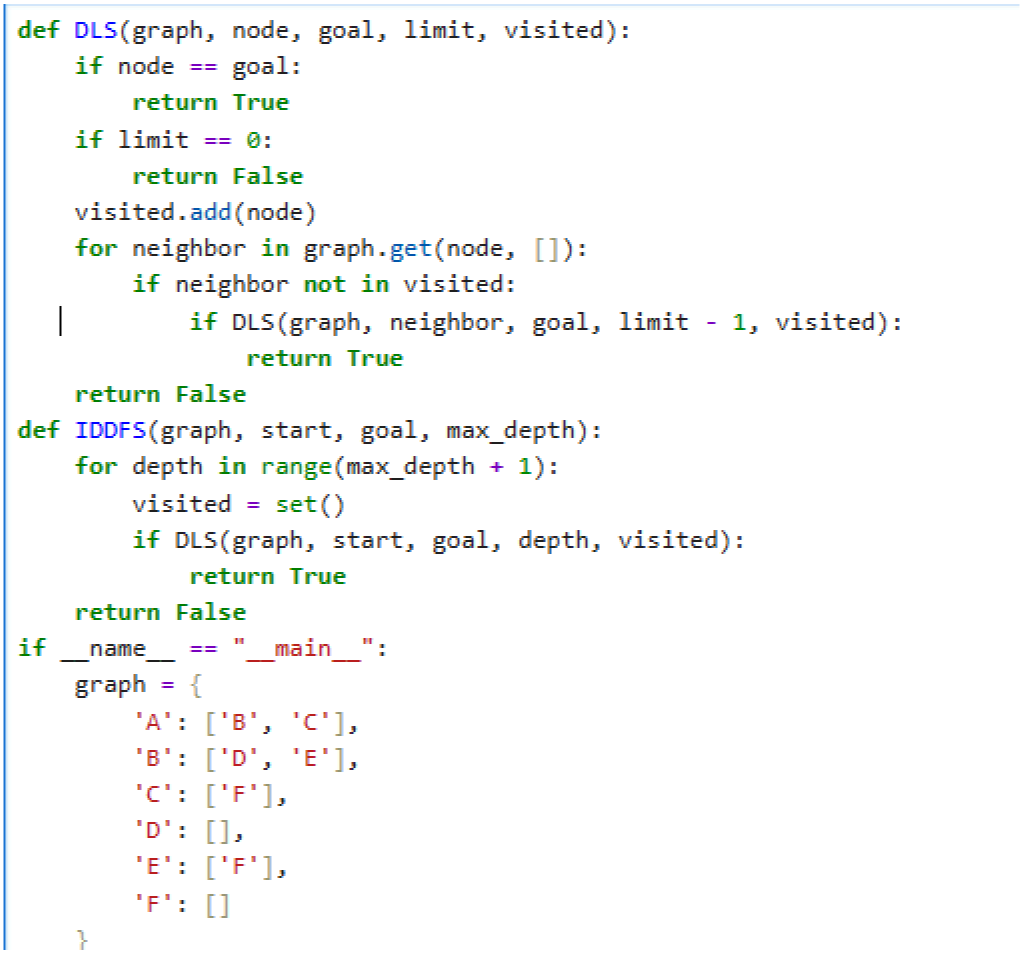
```python
def a_star_manhattan(start, goal):
    open_list = []
    heapq.heappush(open_list, (manhattan_distance(start, goal), 0, start, [start]))
    closed = set()

    while open_list:
        f, g, state, path = heapq.heappop(open_list)

        if state == goal:
            return path

        if state in closed:
            continue
        closed.add(state)

        for neighbor in get_neighbors(state):
            if neighbor not in closed:
                g_new = g + 1
                h_new = manhattan_distance(neighbor, goal)
                f_new = g_new + h_new
                heapq.heappush(open_list, (f_new, g_new, neighbor, path + [neighbor]))

    return None
if __name__ == "__main__":
    start = (1, 2, 3,
             4, 5, 6,
             0, 7, 8)
```



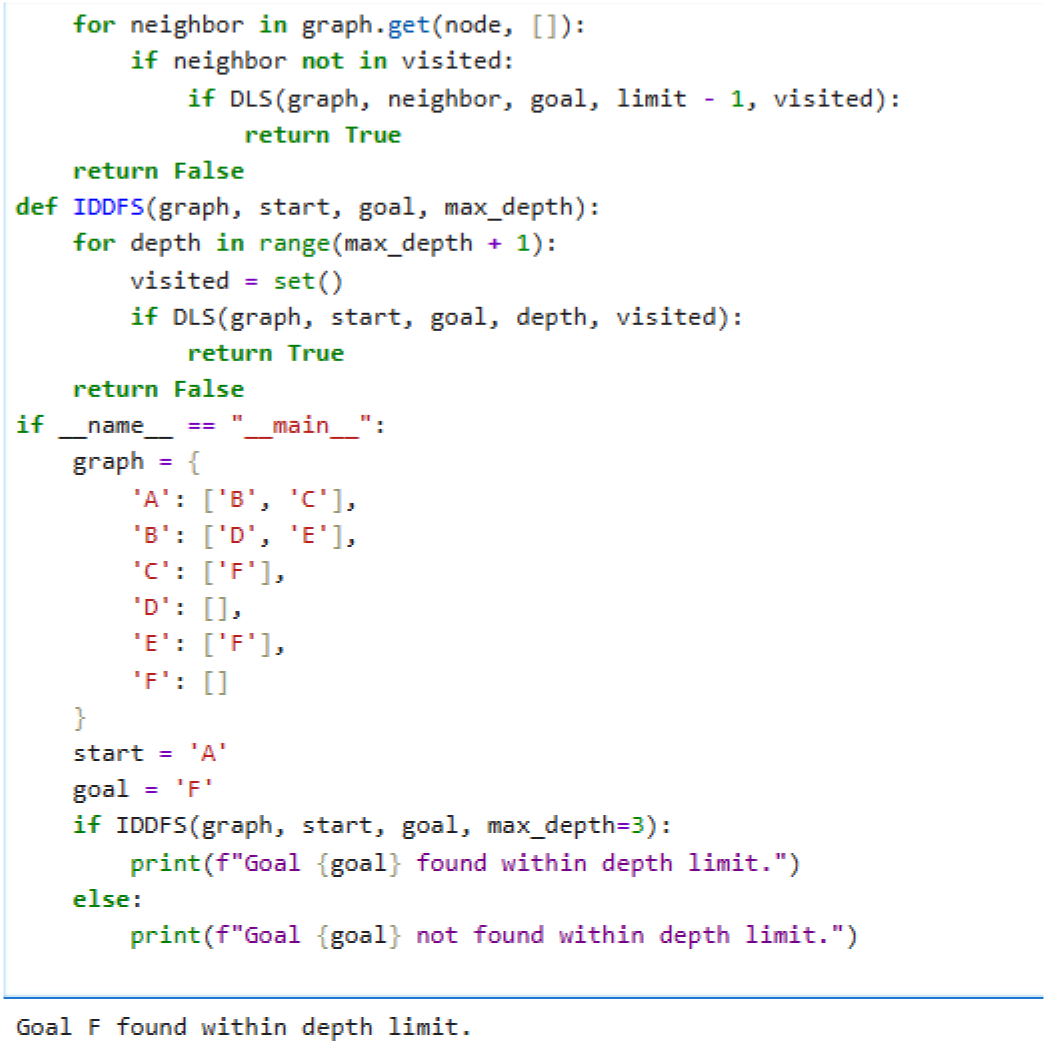
```
    start = (1, 2, 3,
             4, 5, 6,
             0, 7, 8)
goal = (1, 2, 3,
            4, 5, 6,
            7, 8, 0)
    solution = a_star_manhattan(start, goal)
    if solution:
        print("Solution found in", len(solution)-1, "moves.")
        for step in solution:
            for i in range(0, 9, 3):
                print(step[i:i+3])
            print("-----")    else:
        print("No solution found.")
```

```
Solution found in 2 moves.
(1, 2, 3)
(4, 5, 6)
(0, 7, 8)
-----
(1, 2, 3)
(4, 5, 6)
(7, 0, 8)
-----
(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
-----
```

Iterative deepening depth first



```python
def DLS(graph, node, goal, limit, visited):
    if node == goal:
        return True
    if limit == 0:
        return False
    visited.add(node)
    for neighbor in graph.get(node, []):
        if neighbor not in visited:
            if DLS(graph, neighbor, goal, limit - 1, visited):
                return True
    return False
def IDDFS(graph, start, goal, max_depth):
    for depth in range(max_depth + 1):
        visited = set()
        if DLS(graph, start, goal, depth, visited):
            return True
    return False
if __name__ == "__main__":
    graph = {
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': ['F'],
        'D': [],
        'E': ['F'],
        'F': []
    }
```



```
    for neighbor in graph.get(node, []):
        if neighbor not in visited:
            if DLS(graph, neighbor, goal, limit - 1, visited):
                return True
    return False
def IDDFS(graph, start, goal, max_depth):
    for depth in range(max_depth + 1):
        visited = set()
        if DLS(graph, start, goal, depth, visited):
            return True
    return False
if __name__ == "__main__":
    graph = {
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': ['F'],
        'D': [],
        'E': ['F'],
        'F': []
    }
    start = 'A'
    goal = 'F'
    if IDDFS(graph, start, goal, max_depth=3):
        print(f"Goal {goal} found within depth limit.")
    else:
        print(f"Goal {goal} not found within depth limit.")
```

```
Goal F found within depth limit.
```