TASK - 1 | Solve 8- Puzzle problem:

- Consist of 3 × 3 board with eight numbered tiles and a blank space.
- A tile adjacent to the blank space can slide into the space.
- The objective is to reach a specified goal state, such as the one shown in the discussion above.

In [ ]:
```
from collections import deque

goal_state = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, None]]

moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]


def get_blank_pos(state):
    """Function to find the position of the blank space in the puzzle."""
    for i in range(3):
        for j in range(3):
            if state[i][j] is None:
                return i, j


def valid_move(x, y):
    return 0 <= x < 3 and 0 <= y < 3


def bfs(initial_state):
    queue = deque([(initial_state, [])])  # (state, path)
    visited = set()
    print(queue)

    while queue:
        state, path = queue.popleft()
        if state == goal_state:
            return path
        visited.add(tuple(map(tuple, state)))

        # Find the position of the blank space
        blank_x, blank_y = get_blank_pos(state)

        # Generate possible moves
        for dx, dy in moves:
            new_x, new_y = blank_x + dx, blank_y + dy
            if valid_move(new_x, new_y):
                new_state = [row[:] for row in state]  # Deep copy

                # Swap the blank space with the adjacent tile
                new_state[blank_x][blank_y], new_state[new_x][new_y] = new_state[ne

                if tuple(map(tuple, new_state)) not in visited:
```

```
                    queue.append((new_state, path + [(new_x, new_y)]))
                    visited.add(tuple(map(tuple, new_state)))

    return None  # If no solution found


def print_solution(path):
    """Prints the solution path."""
    if path:
        print("Steps to reach the goal state:")
        for step, (x, y) in enumerate(path):
            print(f"Step {step + 1}: Move blank space to position ({x}, {y})")
    else:
        print("No solution found!")

initial_state = [[2, 3, 6],
                 [1, 5, None],
                 [4, 7, 8]]

solution_path = bfs(initial_state)
print_solution(solution_path)
```

```
deque([([[2, 3, 6], [1, 5, None], [4, 7, 8]], [])])
Steps to reach the goal state:
Step 1: Move blank space to position (0, 2)
Step 2: Move blank space to position (0, 1)
Step 3: Move blank space to position (0, 0)
Step 4: Move blank space to position (1, 0)
Step 5: Move blank space to position (2, 0)
Step 6: Move blank space to position (2, 1)
Step 7: Move blank space to position (2, 2)
```

TASK - 2 | Solve depth first search and breadth first search of given graph starting from node A and reaching goal node G:

In [ ]:

```
graph = {
    'A': ['B', 'D'],
    'B': ['C', 'E'],
    'D': ['E', 'G', 'H'],
    'E': ['C', 'F'],
    'G': ['H']
}

def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return []
```

```
def bfs(graph, start, goal):
    queue = [[start]]
    visited = set()

    while queue:
        path = queue.pop(0)
        node = path[-1]
        if node == goal:
            return path
        if node not in visited:
            visited.add(node)
            for neighbor in graph.get(node, []):
                new_path = list(path)
                new_path.append(neighbor)
                queue.append(new_path)

start_node = 'A'
goal_node = 'G'

dfs_path = dfs(graph, start_node, goal_node)
print("Depth-first search path:", dfs_path)


bfs_path = bfs(graph, start_node, goal_node)
print("Breadth-first search path:", bfs_path)
```

```
Depth-first search path: ['A', 'D', 'G']
Breadth-first search path: ['A', 'D', 'G']
```