Lad 1

from collections import deque

def get\_next\_states(state, cap\_a, cap\_b):

    """Generates the next possible states from the current state."""

    a, b = state

    next\_states = []

    actions = [

        ((cap\_a, b), f"Đổ đầy bình A ({cap\_a} lít)") if a < cap\_a else None,

        ((a, cap\_b), f"Đổ đầy bình B ({cap\_b} lít)") if b < cap\_b else None,

        ((0, b), "Đổ cạn bình A") if a > 0 else None,

        ((a, 0), "Đổ cạn bình B") if b > 0 else None,

        ((a - min(a, cap\_b - b), b + min(a, cap\_b - b)), "Đổ từ bình A sang bình B") if a > 0 and b < cap\_b else None,

        ((a + min(b, cap\_a - a), b - min(b, cap\_a - a)), "Đổ từ bình B sang bình A") if b > 0 and a < cap\_a else None

    ]

    next\_states = [act for act in actions if act is not None]

    return next\_states

def solve\_oil\_pouring(cap\_a, cap\_b, target):

    """

    Solves the oil pouring problem using Breadth-First Search (BFS)

    to find the shortest solution and counts the states checked.

    Accepts arbitrary jug capacities and target volume.

    """

    initial\_state = (0, 0)

    queue = deque([(initial\_state, [((0, 0), "Bắt đầu")])])

    visited = {initial\_state}

    states\_checked = 0

    while queue:

        states\_checked += 1

        current\_state, path = queue.popleft()

        a, b = current\_state

        if a == target:

            return path, states\_checked

        for next\_state, action in get\_next\_states(current\_state, cap\_a, cap\_b):

            if next\_state not in visited:

                visited.add(next\_state)

                new\_path = path + [(next\_state, action)]

                queue.append((next\_state, new\_path))

    return None, states\_checked # No solution found

def print\_solution(solution):

    """Prints the steps of a solution."""

    print("🔍 Các bước giải:")

    for index, (state, action) in enumerate(solution):

        a, b = state

        print(f"Bước {index}: Bình A = {a} lít, Bình B = {b} lít | 🛠 Hành động: {action}")

def main():

    """

    Hàm chính giải bài toán đo dầu bằng hai bình với dung tích và mục tiêu tùy ý.

    In ra lời giải ngắn nhất nếu tồn tại và số lượng trạng thái đã kiểm tra.

    """

    # Scenario 1: Default case (5L, 3L, target 4L)

    cap\_a\_default = 5

    cap\_b\_default = 3

    target\_default = 4

    print(f"--- Giải bài toán: Bình {cap\_a\_default}L, Bình {cap\_b\_default}L, Mục tiêu {target\_default}L ---")

    solution\_default, checked\_count\_default = solve\_oil\_pouring(cap\_a\_default, cap\_b\_default, target\_default)

    if solution\_default:

        print(f" Tìm thấy lời giải để đo được {target\_default} lít dầu.")

        print("\n Lời giải ngắn nhất:")

        print\_solution(solution\_default)

        print(f"\nTổng số trạng thái đã kiểm tra: {checked\_count\_default}")

    else:

        print(f" Không thể đo {target\_default} lít dầu với bình {cap\_a\_default} lít và bình {cap\_b\_default} lít.")

        print(f"\nTổng số trạng thái đã kiểm tra: {checked\_count\_default}")

    print("\n" + "="\*50 + "\n") # Separator

    # Scenario 2: New case (7L, 4L, target 6L)

    cap\_a\_new = 7

    cap\_b\_new = 4

    target\_new = 6

    print(f"--- Giải bài toán: Bình {cap\_a\_new}L, Bình {cap\_b\_new}L, Mục tiêu {target\_new}L ---")

    solution\_new, checked\_count\_new = solve\_oil\_pouring(cap\_a\_new, cap\_b\_new, target\_new)

    if solution\_new:

        print(f" Tìm thấy lời giải để đo được {target\_new} lít dầu.")

        print("\n Lời giải ngắn nhất:")

        print\_solution(solution\_new)

        print(f"\nTổng số trạng thái đã kiểm tra: {checked\_count\_new}")

    else:

        print(f" Không thể đo {target\_new} lít dầu với bình {cap\_a\_new} lít và bình {cap\_b\_new} lít.")

        print(f"\nTổng số trạng thái đã kiểm tra: {checked\_count\_new}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

--- Giải bài toán: Bình 5L, Bình 3L, Mục tiêu 4L

--- ✅ Tìm thấy lời giải để đo được 4 lít dầu.

🌟 Lời giải ngắn nhất:

🔍 Các bước giải: Bước 0: Bình A = 0 lít, Bình B = 0 lít | 🛠 Hành động: Bắt đầu

Bước 1: Bình A = 5 lít, Bình B = 0 lít | 🛠 Hành động: Đổ đầy bình A (5 lít)

Bước 2: Bình A = 2 lít, Bình B = 3 lít | 🛠 Hành động: Đổ từ bình A sang bình B

Bước 3: Bình A = 2 lít, Bình B = 0 lít | 🛠 Hành động: Đổ cạn bình B

Bước 4: Bình A = 0 lít, Bình B = 2 lít | 🛠 Hành động: Đổ từ bình A sang bình B Bước 5: Bình A = 5 lít, Bình B = 2 lít | 🛠 Hành động: Đổ đầy bình A (5 lít)

Bước 6: Bình A = 4 lít, Bình B = 3 lít | 🛠 Hành động: Đổ từ bình A sang bình B Tổng số trạng thái đã kiểm tra: 13

**lad2**

import copy

is\_valid\_calls = 0

def find\_empty(board):

    """

    Tìm vị trí ô trống (giá trị bằng 0) trong bảng Sudoku.

    Parameters:

    - board (List[List[int]]): Bảng Sudoku 9x9.

    Returns:

    - Tuple[int, int]: Vị trí (hàng, cột) của ô trống đầu tiên tìm thấy.

    - None: Nếu bảng không còn ô trống.

    """

    for row in range(9):

        for col in range(9):

            if board[row][col] == 0:

                return row, col

    return None

def is\_valid(board, num, pos):

    """

    Kiểm tra xem số `num` có thể được điền vào vị trí `pos` trên bảng Sudoku mà không vi phạm luật.

    Parameters:

    - board (List[List[int]]): Bảng Sudoku 9x9.

    - num (int): Số cần kiểm tra (1-9).

    - pos (Tuple[int, int]): Vị trí cần kiểm tra (row, col).

    Returns:

    - bool: True nếu hợp lệ, False nếu vi phạm luật Sudoku.

    """

    global is\_valid\_calls

    is\_valid\_calls += 1

    row, col = pos

    # Kiểm tra hàng

    if any(board[row][c] == num and c != col for c in range(9)):

        return False

    # Kiểm tra cột

    if any(board[r][col] == num and r != row for r in range(9)):

        return False

    # Kiểm tra ô vuông 3x3

    box\_start\_row = (row // 3) \* 3

    box\_start\_col = (col // 3) \* 3

    for r in range(box\_start\_row, box\_start\_row + 3):

        for c in range(box\_start\_col, box\_start\_col + 3):

            if board[r][c] == num and (r, c) != pos:

                return False

    return True

def is\_initial\_board\_valid(board):

    """

    Kiểm tra xem bảng Sudoku ban đầu có hợp lệ không.

    Parameters:

    - board (List[List[int]]): Bảng Sudoku 9x9.

    Returns:

    - bool: True nếu hợp lệ, False nếu không.

    """

    # Create a temporary copy to avoid modifying the original board during validation

    temp\_board = copy.deepcopy(board)

    for row in range(9):

        for col in range(9):

            if temp\_board[row][col] != 0:

                num = temp\_board[row][col]

                temp\_board[row][col] = 0 # Temporarily set to 0 to check validity of this number

                if not is\_valid(temp\_board, num, (row, col)):

                    return False

                temp\_board[row][col] = num # Restore the value

    return True

def solve\_sudoku(board, solutions):

    """

    Giải bảng Sudoku bằng phương pháp đệ quy và quay lui, lưu trữ tất cả các lời giải.

    Parameters:

    - board (List[List[int]]): Bảng Sudoku 9x9 với ô trống là số 0.

    - solutions (List[List[List[int]]]): Danh sách để lưu trữ các lời giải.

    Returns:

    - bool: Luôn trả về True để tiếp tục tìm kiếm các lời giải khác.

    """

    empty = find\_empty(board)  # Tìm ô chưa điền (giá trị = 0)

    if not empty:

        solutions.append(copy.deepcopy(board)) # Lưu trữ một bản sao của lời giải

        return True  # Tìm thấy một lời giải, tiếp tục tìm kiếm các lời giải khác

    row, col = empty

    for num in range(1, 10):  # Thử các số từ 1 đến 9

        if is\_valid(board, num, (row, col)):

            board[row][col] = num  # Gán số hợp lệ

            solve\_sudoku(board, solutions)  # Đệ quy tiếp

            board[row][col] = 0  # Quay lui sau khi duyệt xong một nhánh

    return True # Luôn trả về True để tiếp tục tìm kiếm các lời giải khác

def print\_board(board):

    """

    In bảng Sudoku theo định dạng trực quan, chia cột và hàng thành từng khối 3x3.

    Parameters:

    - board (List[List[int]]): Bảng Sudoku 9x9, với giá trị 0 đại diện cho ô trống.

    """

    for i in range(9):

        if i % 3 == 0 and i != 0:

            print("------+-------+------")

        for j in range(9):

            if j % 3 == 0 and j != 0:

                print("| ", end="")

            cell = board[i][j]

            print(cell if cell != 0 else ".", end=" ")

        print()  # Xuống dòng sau mỗi hàng

def main():

    """

    Hàm chính khởi tạo bảng Sudoku, in bảng ban đầu,

    giải bằng thuật toán đệ quy, và in kết quả.

    """

    global is\_valid\_calls

    is\_valid\_calls = 0  # Reset the counter at the start of main

    # 🌟 Bảng Sudoku mẫu với một số ô đã điền

    board = [

        [5, 3, 0, 0, 7, 0, 0, 0, 0],

        [6, 0, 0, 1, 9, 5, 0, 0, 0],

        [0, 9, 8, 0, 0, 0, 0, 6, 0],

        [8, 0, 0, 0, 6, 0, 0, 0, 3],

        [4, 0, 0, 8, 0, 3, 0, 0, 1],

        [7, 0, 0, 0, 2, 0, 0, 0, 6],

        [0, 6, 0, 0, 0, 0, 2, 8, 0],

        [0, 0, 0, 4, 1, 9, 0, 0, 5],

        [0, 0, 0, 0, 8, 0, 0, 7, 9]

    ]

    print("🧩 Bảng Sudoku ban đầu:")

    print\_board(board)

    if not is\_initial\_board\_valid(board):

        print("\n Bảng Sudoku ban đầu không hợp lệ.")

        print(f"Tổng số lần gọi is\_valid: {is\_valid\_calls}")

        return

    solutions = []

    solve\_sudoku(board, solutions)

    if solutions:

        print(f"\n Tìm thấy {len(solutions)} lời giải cho bảng Sudoku!")

        for i, solution in enumerate(solutions):

            print(f"\nLời giải {i + 1}:")

            print\_board(solution)

    else:

        print("\n Không tìm thấy lời giải cho bảng Sudoku.")

    print(f"\nTổng số lần gọi is\_valid: {is\_valid\_calls}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

🧩 Bảng Sudoku ban đầu:

5 3 . | . 7 . | . . .

6 . . | 1 9 5 | . . .

. 9 8 | . . . | . 6 .

------+-------+------

8 . . | . 6 . | . . 3 4

. . | 8 . 3 | . . 1 7

. . | . 2 . | . . 6

------+-------+------

. 6 . | . . . | 2 8 .

. . . | 4 1 9 | . . 5

. . . | . 8 . | . 7 9

✅ Tìm thấy 1 lời giải cho bảng Sudoku!

🌟 Lời giải 1:

5 3 4 | 6 7 8 | 9 1 2

6 7 2 | 1 9 5 | 3 4 8

1 9 8 | 3 4 2 | 5 6 7

------+-------+------

8 5 9 | 7 6 1 | 4 2 3

4 2 6 | 8 5 3 | 7 9 1

7 1 3 | 9 2 4 | 8 5 6

------+-------+------

9 6 1 | 5 3 7 | 2 8 4

2 8 7 | 4 1 9 | 6 3 5

3 4 5 | 2 8 6 | 1 7 9

Tổng số lần gọi is\_valid: 41709

**Lad 3**

def solve\_pathfinding(grid, start, goal, allow\_diagonal=False):

    m, n = len(grid), len(grid[0])

    normalized = [[0 if cell in ['S', 'G'] else cell for cell in row] for row in grid]

    visited = {start}

    path = [(start, "Bắt đầu")]

    state\_counter = [0]

    solution = []

    directions = [(-1, 0, "Lên"), (1, 0, "Xuống"), (0, -1, "Trái"), (0, 1, "Phải")]

    if allow\_diagonal:

        directions += [(-1, -1, "Trên-Trái"), (-1, 1, "Trên-Phải"),

                       (1, -1, "Dưới-Trái"), (1, 1, "Dưới-Phải")]

    def dfs(pos):

        if pos == goal:

            solution.extend(path)

            return True

        state\_counter[0] += 1

        r, c = pos

        for dr, dc, move in directions:

            nr, nc = r + dr, c + dc

            if 0 <= nr < m and 0 <= nc < n and normalized[nr][nc] != 1 and (nr, nc) not in visited:

                visited.add((nr, nc))

                path.append(((nr, nc), move))

                if dfs((nr, nc)):

                    return True

                path.pop()

                visited.remove((nr, nc))

        return False

    dfs(start)

    return solution, state\_counter[0]

def print\_grid(grid, path=None):

    m, n = len(grid), len(grid[0])

    display = [[grid[i][j] for j in range(n)] for i in range(m)]

    if path:

        for (r, c), \_ in path[1:]:  # bỏ bước đầu

            if display[r][c] not in ['S', 'G']:

                display[r][c] = '\*'

    for row in display:

        print(" ".join(f"{cell:2}" for cell in row))

    print()

def print\_solution(solution):

    print(" Hành trình robot:")

    for i, ((r, c), move) in enumerate(solution):

        print(f"Bước {i}: {move} ➝ ({r}, {c})")

def main():

    grid = [

        ['S', 0, 0, 1],

        [0, 1, 0, 0],

        [0, 0, 1, 0],

        [1, 0, 0, 'G']

    ]

    start, goal = (0, 0), (3, 3)

    print(" Bản đồ lưới:")

    print\_grid(grid)

    print(" Robot di chuyển theo 4 hướng:")

    sol4, count4 = solve\_pathfinding(grid, start, goal, allow\_diagonal=False)

    print\_solution(sol4)

    print(f"Số trạng thái đã duyệt: {count4}")

    print\_grid(grid, sol4)

    print(" Robot di chuyển theo 8 hướng:")

    sol8, count8 = solve\_pathfinding(grid, start, goal, allow\_diagonal=True)

    print\_solution(sol8)

    print(f"Số trạng thái đã duyệt: {count8}")

    print\_grid(grid, sol8)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Bản đồ lưới:

S 0 0 1

0 1 0 0

0 0 1 0

1 0 0 G

Robot di chuyển theo 4 hướng:

Hành trình robot:

Bước 0: Bắt đầu ➝ (0, 0)

Bước 1: Xuống ➝ (1, 0)

Bước 2: Xuống ➝ (2, 0)

Bước 3: Phải ➝ (2, 1)

Bước 4: Xuống ➝ (3, 1)

Bước 5: Phải ➝ (3, 2)

Bước 6: Phải ➝ (3, 3)

Số trạng thái đã duyệt: 6

S 0 0 1

\* 1 0 0

\* \* 1 0

1 \* \* G

Robot di chuyển theo 8 hướng:

Hành trình robot:

Bước 0: Bắt đầu ➝ (0, 0)

Bước 1: Xuống ➝ (1, 0)

Bước 2: Xuống ➝ (2, 0)

Bước 3: Phải ➝ (2, 1)

Bước 4: Xuống ➝ (3, 1)

Bước 5: Phải ➝ (3, 2)

Bước 6: Phải ➝ (3, 3)

Số trạng thái đã duyệt: 6

S 0 0 1

\* 1 0 0

\* \* 1 0

1 \* \* G