```
import sudoku_board
import numpy as np
import matplotlib.pyplot as plt
from typing import Tuple, Set, Dict


def PrepBoard(path: str = r"A2\sudoku_small.csv", n: int = 1) -> Tuple["np.ndarray[np.int8]",
                                                                       Set[Tuple[Tuple[int, int],
                                                                                 Tuple[int, int
                                                                                 ]]],
                                                                       Dict[Tuple[int, int],
                                                                            Set[int]]]:
    """
    Loads the nth board, generrates the constraints and domains

    Args:
        board: Board to be printed
        empty_value: Values to be printed if cell is empty
    Print:
        NO_SOL => There is no solution
        FAIL => There is a implementation error
        SOLVED => The board has been fully solved
        PARTIAL_SOLVE => The board has been partial solved
    """
    board = sudoku_board.load_file(path=path, n=n)[-1]
    constraints = sudoku_board.create_constraint_set()
    domains = sudoku_board.create_domain_set(board)
    return board, constraints, domains


def testBoard(n: int = 1) -> None:
    """
    Prints the nth board, applies AC-3 to nth board, prints the board and

    Args:
        board: Board to be printed
        empty_value: Values to be printed if cell is empty
    Print:
        NO_SOL => There is no solution
        FAIL => There is a implementation error
        SOLVED => The board has been fully solved
        PARTIAL_SOLVE => The board has been partial solved
    """
    pre, constraints, domains = PrepBoard(n)
    post_Domains = sudoku_board.AC3(constraints, domains)

    print(f"Sudoku Board #{n}")
    sudoku_board.pretty_print_board(pre)

    if not post_Domains:
        print("NO_SOL: AC-3 has identified board #{n} has no solution")
    elif not sudoku_board.validSolve(pre, sudoku_board.domains2Board(post_Domains)):
        print("FAIL: AC-3 has been implemented incorrectly")
        for key, val in post_Domains.items():
            print(f"{key} | {val}")
    elif sudoku_board.solved(post_Domains):
        print("SOLVED: AC-3 has fully solved the board")
        sudoku_board.pretty_print_board(sudoku_board.domains2Board(post_Domains))
    else:
        print("PARTIAL_SOLVE: AC-3 has partially solved the board")

        for key, val in post_Domains.items():
            print(f"{key} | {val}")

        post_Domains = sudoku_board.backtracking_search(post_Domains)

        if not post_Domains:
            print("NO_SOL: BackTrack has identified board #{n} has no solution")
        elif not sudoku_board.validSolve(pre, sudoku_board.domains2Board(post_Domains)):
            print("FAIL: BackTrack has been implemented incorrectly")
            for key, val in post_Domains.items():
                print(f"{key} | {val}")
        elif sudoku_board.solved(post_Domains):
            print("SOLVED: BackTrack has fully solved the board")
            sudoku_board.pretty_print_board(sudoku_board.domains2Board(post_Domains))
        else:
            print("NO_SOL: BackTrack has partially solved the board")


def solve(pre: "np.ndarray[np.int8]")-> bool:
    """
    Takes in a board and tries to solve it using AC-3 and BackTracking

    Args:
        board: Board to be printed
    Print:
        Prints out the solved board or which algorithm failed and the domain set of the failed board
    """

    constraints = sudoku_board.create_constraint_set()
    domains = sudoku_board.create_domain_set(pre)

    post_Domains, qlen = sudoku_board.AC3(constraints, domains, True)

    plt.plot(qlen)
    plt.xlabel("Step #")
    plt.ylabel("Queue Length")
    plt.savefig("queue_length")

    if not post_Domains:
        print("Board is not vaild")
        return False
    elif sudoku_board.solved(post_Domains):
        print("Board solved with AC3")
        sudoku_board.pretty_print_domain(post_Domains)
        return True
    elif not sudoku_board.validDomains(post_Domains):
        print("AC-3 has created an invaild board")
        for key, val in post_Domains:
            print(f"{key} | {val}")
        raise ValueError("AC-3 has created an invaild board")
    else:
        print("AC-3 did not solve the board. Attepting to solve with backtracking...")
        post_Domains = sudoku_board.backtracking_search(domains)
        if not post_Domains:
            print("Board is not vaild")
            return False
        elif not sudoku_board.solved(post_Domains):
            print("Backtracking has not solved the board")
            for key, val in post_Domains:
                print(f"{key} | {val}")
            raise ValueError("Backtracking has not solved the board")
        elif not sudoku_board.validDomains(post_Domains):
            print("AC-3 has created an invaild board")
            for key, val in post_Domains:
                print(f"{key} | {val}")
            raise ValueError("AC-3 has created an invaild board")
        else:
            print("Solved with backtracking:")
            sudoku_board.pretty_print_domain(post_Domains)


#solve(sudoku_board.load_file(path =r"A2\sudoku_small.csv",n=1)[-1])

for i in range(98,102):
    solve(sudoku_board.load_file(path =r"A2\sudoku_small.csv",n=i)[-1])
```