Sudoku testing

https://youtube.com/playlist?list=PLAwxTw4SYaPkWVHeC_8aSIbSxE_NXI76g&si=lhs0mKJKm5pW3D1R

1. Sudoku checker

```
# SPECIFICATION:
#
# check_sudoku() determines whether its argument is a valid Sudoku
# grid. It can handle grids that are completely filled in, and also
# grids that hold some empty cells where the player has not yet
# written numbers.
#
# First, the code must do some sanity checking to make sure that its
# argument:
#
# - is a 9x9 list of lists
#
# - contains, in each of its 81 elements, an integer in the range 0..9
#
# If either of these properties does not hold, check_sudoku must
# return None.
#
# If the sanity checks pass, the code should return True if all of
# the following hold, and False otherwise:
#
# - each number in the range 1..9 occurs only once in each row
#
# - each number in the range 1..9 occurs only once in each column
#
# - each number the range 1..9 occurs only once in each of the nine
#   3x3 sub-grids, or "boxes", that make up the board
#
# This diagram (which depicts a valid Sudoku grid) illustrates how the
# grid is divided into sub-grids:
#
# 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 | 0 0 0
# 2 8 7 | 4 1 9 | 0 0 0
# 3 4 5 | 2 8 6 | 0 0 0
#
# Please keep in mind that a valid grid (i.e., one for which the
# function returns True) may contain 0 multiple times in a row,
# column, or sub-grid. Here we are using 0 to represent an element of
# the Sudoku grid that the player has not yet filled in.
```

```
# check_sudoku should return None
ill_formed = [[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],  # <---
              [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]]

# check_sudoku should return True
valid = [[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]]

# check_sudoku should return False
invalid = [[5,3,4,6,7,8,9,1,2],
           [6,7,2,1,9,5,3,4,8],
           [1,9,8,3,8,2,5,6,7],
           [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]]

# check_sudoku should return True
easy = [[2,9,0,0,0,0,0,7,0],
        [3,0,6,0,0,8,4,0,0],
        [8,0,0,0,4,0,0,0,2],
        [0,2,0,0,3,1,0,0,7],
        [0,0,0,0,8,0,0,0,0],
        [1,0,0,9,5,0,0,6,0],
        [7,0,0,0,9,0,0,0,1],
        [0,0,1,2,0,0,3,0,6],
        [0,3,0,0,0,0,0,5,9]]

# check_sudoku should return True
hard = [[1,0,0,0,0,7,0,9,0],
        [0,3,0,0,2,0,0,0,8],
        [0,0,9,6,0,0,5,0,0],
        [0,0,5,3,0,0,9,0,0],
        [0,1,0,0,8,0,0,0,2],
        [6,0,0,0,0,4,0,0,0],
        [3,0,0,0,0,0,0,1,0],
        [0,4,0,0,0,0,0,0,7],
        [0,0,7,0,0,0,3,0,0]]
```

- A non-clever solution to get the output for the 5 test cases given is presented below. There are issues with this code. How can we break it using a random fuzzer?

```
def check_sudoku(grid):
    if type(grid) is not list or len(grid) != 9:
        return None

    for row in range(0,9):
        if type(grid[row]) is not list or len(grid[row]) != 9:
            return None
        for col in range(0,9):
            if (type(grid[row][col]) is not int or
                grid[row][col] < 0 or
                    grid[row][col] > 9):
                return None

    # check rows
    for row in range(0,9):
        d = {}
        for col in range(0,9):
            val = grid[row][col]
            if val != 0 and val in d:
                return False
            d[val] = 1

    # check columns
    for col in range(0,9):
        d = {}
        for row in range(0,9):
            val = grid[row][col]
            if val != 0 and val in d:
                return False
            d[val] = 1

    # check sub-grids
    for i in range(0,3):
        for j in range(0,3):
            d = {}
            for ii in range(0,3):
                for jj in range(0,3):
                    row = (3 * i) + ii
                    col = (3 * j) + jj
                    val = grid[row][col]
                    if val != 0 and val in d:
                        return False
                    d[val] = 1

    return True

print(check_sudoku(ill_formed))  # --> None
print(check_sudoku(valid))      # --> True
print(check_sudoku(invalid))    # --> False
print(check_sudoku(easy))       # --> True
print(check_sudoku(hard))       # --> True
```

2. Sudoku solver

```
# CHALLENGE PROBLEM:
#
# We use check_sudoku function as the basis for solve_sudoku(): a
# function that takes a partially-completed Sudoku grid and replaces
# each 0 cell with an integer in the range 1..9 in such a way that the
# final grid is valid.
#
# There are many ways to cleverly solve a partially-completed Sudoku
# puzzle, but a brute-force recursive solution with backtracking is a
# perfectly good option. The solver should return None for broken
# input, False for inputs that have no valid solutions, and a valid
# 9x9 Sudoku grid containing no 0 elements otherwise. In general, a
# partially-completed Sudoku grid does not have a unique solution. We
# should just return some member of the set of solutions.
#
# A solve_sudoku() in this style can be implemented in about 16 lines
# without making any particular effort to write concise code.

# solve_sudoku should return None
ill_formed = [[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],  # <---
              [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]]

# solve_sudoku should return valid unchanged
valid = [[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]]

# solve_sudoku should return False
invalid = [[5,3,4,6,7,8,9,1,2],
           [6,7,2,1,9,5,3,4,8],
           [1,9,8,3,8,2,5,6,7],
           [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]]
```

```
# solve_sudoku should return a sudoku grid which passes a sudoku checker.
# There may be multiple correct grids which can be made from this starting
# grid.
easy = [[2,9,0,0,0,0,0,7,0],
        [3,0,6,0,0,8,4,0,0],
        [8,0,0,0,4,0,0,0,2],
        [0,2,0,0,3,1,0,0,7],
        [0,0,0,0,8,0,0,0,0],
        [1,0,0,9,5,0,0,6,0],
        [7,0,0,0,9,0,0,0,1],
        [0,0,1,2,0,0,3,0,6],
        [0,3,0,0,0,0,0,5,9]]
#
hard = [[1,0,0,0,0,7,0,9,0],
        [0,3,0,0,2,0,0,0,8],
        [0,0,9,6,0,0,5,0,0],
        [0,0,5,3,0,0,9,0,0],
        [0,1,0,0,8,0,0,0,2],
        [6,0,0,0,0,4,0,0,0],
        [3,0,0,0,0,0,0,1,0],
        [0,4,0,0,0,0,0,0,7],
        [0,0,7,0,0,0,3,0,0]]
```

- A brute-force algorithm using check_sudoku function presented in section 1:

```
import copy
import sys

def solve_sudoku(grid):
    res = check_sudoku(grid)
    if res is None or res is False:
        return res

    grid = copy.deepcopy(grid)

    # find the first 0 element and change it to each of 1..9
    # recursively calling this function on the result
    for row in range(0,9):
        for col in range(0,9):
            if grid[row][col] == 0:
                for n in range(1,10):
                    grid[row][col] = n
                    new = solve_sudoku(grid)
                    if new is not False:
                        return new
                # backtrack
                return False

    # if we get here, we found no zeros and so we're finished
    return grid

print(solve_sudoku(ill_formed))  # --> None
print(solve_sudoku(valid))      # --> True
print(solve_sudoku(invalid))    # --> False
print(solve_sudoku(easy))       # --> True
print(solve_sudoku(hard))       # --> True
```

Notă[1]

[1] © Copyright 2022 Lect. dr. Sorina-Nicoleta PREDUT
Toate drepturile rezervate.