PROBLEM Sudoku Puzzle:

A Sudoku board consists of 81 squares (see textbook section 6.2.6), some of which are initially filled with digits from 1 to 9. The puzzle is to fill in all the remaining squares such that no digit appears twice in any row, column, or 3×3 box. A row, column, or box is called a **unit**.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Figure 6.4 (a) A Sudoku puzzle and (b) its solution.

CSP Formulation:

Whenever we formulate a problem into a CSP, we need to identify the following: set of variables *Xi*, set of domains *Di*, and set of constraints *C*:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

All sudoku puzzles can be formulated as CSP by considering each **cell** as a variable. The initial domain of all cells is {1, 2, 3, 4, 5, 6, 7, 8, 9}. The constraints are formulated by the fact that in the solution of a sudoku puzzle, no two cells in a row, column or block can have identical numbers.

Implementation Activities:

The Sudoku puzzle is a classic constraint satisfaction problem (CSP). Implement the Arc-Consistency 3 (AC-3) Algorithm and the Backtracking Algorithm to solve the Sudoku puzzle in Python, and compare their time complexities. The Sudoku puzzle board should feature a graphical user interface (GUI) using Tkinter. The dataset for the Sudoku puzzles is provided.

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
      (Xi, Xj) ← POP(queue)
      if REVISE(csp, Xi, Xj) then
          if size of Di = 0 then return false
          for each Xk in Xi.NEIGHBORS − {Xj} do
            add(Xk, Xi) to queue
  return true

function REVISE(csp, Xi, Xj) returns true iff we revise the domain of Xi
  revised ← false
  for each x in Di do
      if no value y in Dj allows (x, y) to satisfy the constraint between Xi and Xj then
        delete x from Di
        revised ← true
  return revised

function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({}, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
            add inferences to assignment
            result ← BACKTRACK(assignment, csp)
            if result ≠ failure then
              return resultremove {var = value} and inferences from assignment
  return failure
```

Implementation Guidelines:

1. Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 4th edition. Pearson, 2022.
2. Code for the book "Artificial Intelligence: A Modern Approach" https://github.com/aimacode/aima-python (accessed April 15, 2024).
3. Wei-Meng Lee, "Programming Sudoku" www.apress.com/9781590596623 (accessed April 15, 2024).
4. Graphical User Interfaces, http://newcoder.io/gui/ (accessed April 15, 2024).