Sudoku as CSP

See slides for sample board

"""" problem formulation

\[ \begin{align*}
\text{Vars: } & A_1, A_2, \ldots, A_9 \quad \text{ } \quad A_1 = 0, A_2 = 1 \ldots, A_9 = 8 \\
& B_1, B_2, \ldots, B_9 \quad \quad \quad 9, 10, \ldots, 17 \\
& I_1, I_2, \ldots, I_9 \quad 72, 73, \ldots, 80 \\
\end{align*} \]

\[ \text{Instead of 81 separate variables, just refer to each with a number in } [0, 80]. \]

\[ \begin{align*}
\text{Domains: } & A_1 = 0 \text{ (not set) } 9, 1, \ldots, 9^2 \\
& B_1 = 9 \text{ (set) } 9^2, 9^3 \\
\end{align*} \]

\[ \begin{align*}
\text{Constraints: } & \text{AllDiff}(0, 1, \ldots, 8) \text{ (row 1)} \\
& \text{AllDiff} \text{ (row 2)} \\
& \text{AllDiff} \text{ (row 9)} \\
& \text{AllDiff}(0, 9, \ldots, 72) \text{ (col 1)} \\
& \text{AllDiff}(\text{col } 9) \\
& \text{AllDiff}(0, 1, 2, 9, 10, 11, 18, 19, 20) \text{ (box 1)} \\
& \text{AllDiff} \text{ (box 9)} \\
\end{align*} \]

Eventually, will add each pair is an arc of Q.
To Do: Implement back-tracking search with AC3 (forward-checking not required).

initial board

\[ A_1 = 1 \]

Run AC3().
If all arcs consistent, try \( B_1 = 1 \)
if not, back-track

Data Structures:

1. \( \text{vals}[?] := \) \( 9 \times 9 \) int array with initial board values (0 if empty)
   - ex: \( \text{vals}[0][0] \) (A1) = 0
   \( \text{vals}[7][0] \) (B1) = 9

To Do: Store final board values.

2. \( \text{globalDomains}[] := \) array of size \( 81 \) of ArrayLists of Integers

To Do: Store domains of each cell:

ex: \( D_0 = \{1, 2, \ldots, 9\} \), \( D_2 = \{3\} \) (cell A3 already set to 3).
3) neighbors - array of size 81 of ArrayLists of Integers.

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>...</th>
<th>I9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>...</td>
<td>80</td>
</tr>
</tbody>
</table>

To Do: Store neighbors (vars in a constraint) of each var.
ex: neighbors of 0 (Variables in the same row, column, or box)

\[
\begin{align*}
\text{same row as 0} & \quad \text{col} \\
\text{box} & \quad \text{Think about how to not do this manually}
\end{align*}
\]

4) Arc - object with variables \( x_i, x_j \).
   - Represents arc/constraint \( (x_i, x_j) \).

5) globalQueue - Queue of arcs
To Do: Fill with all arcs.
ex: \( (0, 1), (0, 2), \ldots (0, 8) \) \( \geq \) constraints
\( (1, 0), (1, 1), \ldots (1, 8) \} \) \( \geq \) among
\( (8, 0), (8, 1), \ldots (8, 8) \} \) \( \geq \) rows
\( (0, 9), (0, 18), \ldots (0, 72) \} \( \geq \) constraint
\( (1, 10), (1, 19), \ldots (1, 73) \} \( \geq \) among cols
\( (1, 1) \)
$(0,1), (0,2), (0,9), \ldots, (10,20)$ \quad \{ \text{constraints among} \\
$(3,4), (3,5), (3,12), \ldots, (3,23)$ \quad \{ \text{boxes} \}
Methods:

1. AC3_Init():
   - set up globalDomains
   - call allDiff() (see below)
   - call backtrack(0, globalDomains) ← already in code
   - set final board values in vals

2. allDiff():
   (called by AC3_Init())
   - fills neighbors, globalQueue (using binary constraints)

3. backtrack (performs backtracking search with AC3):
   \[ \text{(int cell, GlobalDomains)} \]
   - tries a value for cell (using globalDomains)
   - skip (calls AC3() to check for arc consistencies)
   - if not: backtrack to try another value
   - if yes: call backtrack on next cell

4. AC3 (runs AC3 algorithm)
   - calls Revise() to update domains
   - returns true if consistent; o/w false
// Finds a consistent value for cellnum
backtrack(cellnum, gd)

1. First, check if cellnum already assigned an initial value.
   if (vals[cellnum] != 0)
     // call backtrack on next cell
     backtrack(cellnum + 1, gd).

2. // return false if AC3 detects current assignment not consistent
   if (AC3 returns false)
     return false.
   else? // find a value for this cellnum
     for some value v in cellnum's domain:
       try assigning cellnum = v
       (How do we "try"? Just assign v to cellnum and see if it works)
       with future assignments!

       call backtrack() on next cell (cellnum+1)
       if solution found (backtrack() returns true)
       return true
   else?
     try another value
If we got really lucky...

\[
\begin{align*}
\text{init} & \\
0 &= 1 \\
\text{AC3} &\checkmark \\
1 &= 5 \\
\text{AC3} &\checkmark \\
2 &= 7 \\
\text{AC3} &\checkmark \\
\vdots \\
80 &= 9 \\
\text{AC3} &\checkmark \text{ return true}
\end{align*}
\]

For each variable, we would assign it a value.
- AC3 would find no conflicts
- backtrack would return true.

What is more likely...

\[
\begin{align*}
\text{init} & \\
0 &= 1 \\
\text{AC3} &\checkmark \\
1 &= 5 \\
\text{AC3} &\checkmark \\
2 &= 7 \\
\text{AC3} &\checkmark \text{ returns false}
\end{align*}
\]

backtrack to previous assignment