Sudoku as CSP

See slides for sample board

" " " problem formulation

Vars: \( A_1, A_2, \ldots, A_9 \) \( A_1 = 0, A_2 = 1 \ldots, A_9 = 8 \)
\( B_1, B_2, \ldots, B_9 \) \( 9, 10, \ldots, 17 \)
\( I_1, I_2, \ldots, I_9 \) \( 72, 73, \ldots, 80 \)

Instead of 81 separate
variables, just refer to
each with a number in \([0,80]\)

Domains: \( A_1 = 0 \) (not set) \( 1, 2, \ldots, 9 \)
\( B_1 = 9 \) (set) \( 9, 9, \ldots, 9 \)

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

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

Initial board

\[ A1 = 1, \overline{A1} = 2 \]

Run AC3().
If all arcs consistent, try B1 = 1.
If not, backtrack.

Data Structures:
1. \text{vals}[][][9x9] int array with initial board values (0 if empty).
   - \text{ex: vals}[0][0][7] (A1) = 0
   - \text{vals}[1][0][7] (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:

\[ n \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \}
\[ D_n = \{3\} \text{ (n+11 A1 already set to 3) } \]
(3) neighbors - array of size 81 of ArrayLists of Integers.

```
A1 A2 ... I9
|   |   |   |
0 1 2 ... 80
```

To Do: Store neighbors (vars in a constraint) of each var.

**Example:** neighbors of 0 ? (Variables in the same row, column, or box)

```
1 2 3 
5 6 7 
8 9 10
```

same row as 0

```
1 2 3 
5 6 7 
8 9 10
```

"col"

```
1 2 3 
5 6 7 
8 9 10
```

"box"

Think about how to not do this manually.

(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.

**Example:**

```
(0,1) (0,2), ... (0,8) constraints Note: no (0,0), (1,0) (1,2), ... (1,8) among (1,1), (2,2), etc...
(8,0) (8,1), ... (8,7) rows
```

```
(0,9) (0,18), ... (0,72) constraints
(1,10) (1,19), ... (1,73) among cols
```
(0,1) (0,2) (0,9) ... (0,20) \{ \text{constraints among} \\
(3,4) (3,5) (3,12) \ldots (3,23) \} \text{ boxes}
Methods:

1. AC3_Init():
   - set up globalDomains
   - call allDiff() (see below)
   - call backtrack (\( O \), 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)
   - \( L_i \) (int cell, globalDomains)
   - tries a value for cell (using globalDomains)
   - call revise() to check for arc consistency
   - 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
backtrack(cellnum, gD)

(4) When do we know a solution is found?
   if (cellnum > 80), return true

1. First //check if cellnum already assigned an initial value.
   if (vals[7][I] for cellnum != 0)
      // call backtrack on next cell
      backtrack(cellnum + 1, gD).

(2) //check if previous cellnum assignment is inconsistent
   * if (AC3() finds empty domain)
     return false
   else // find a value for this cellnum for some value v in cellnum's domain
     for each v in cellnum's domain

     // try assigning cellnum to v
     - extract row, col from cellnum
     - vals[row][col] = v
     - Set gD[cellnum] = v
     - call backtrack(cellnum + 1, gD)
     4. if backtrack returns true
        // assigning cellnum to v was consistent
        return true
     else // For cellnum + 1, AC3 returned false
     (see lines...
If we got really lucky...

- init returns true
- For each variable, we would assign it a value
- AC3 would find no conflicts on each call to backtrack().
- returns true
- initial call to backtrack()
- returns true
- We have a satis. assignment

What is more likely...

- AC3 X \rightarrow returns false
- backtrack to previous assignment

*Note:* inconsistent assignment for variable 2 is found not when we assign variable 2 but when we try to assign variable 3!
So call to backtrack(cellnum) detects inconsistency of cellnum-1.
Part 2: Implement heuristic

- Most constrained variable
- Least constraining value
- Forward checking
- Combination
- Your own!

I write in custom solver()