**Formal Definition**

* What is X, D, C
  + X – set of 81 variables {A1,,,A9,,,,,,,,,I1,,,,,,I9}
  + Di for all Xi = {1, 2, 3, 4 ,5, 6, 7, 8, 9}
  + C = 27 AllDiff constraints??

**Implementation Details**

* We have to take the input data and set the Di’s accordingly
* Choosing the Most Constrained variable –
  + Choosing the variable with smallest domain
  + How do we choose the most constraining variable (for tie breaking) ?
    - Choose the variable (i.e cell) with maximum degree – i.e the cell with the most number of unassigned cells that it can constrain
      * Need a method to find if a cell is assigned or not
      * Need a method that returns all neighbors of a cell
  + Least Constraining value
    - While choosing the value of a particular cell, choose that value that rules out fewest values for its peers
      * Need a method that takes (variable, value) – and returns the updated domains of all its neighbors
        + This method can be used for Forward checking
      * Using the above method we can find the value assignment that caused the least no of reductions in domains of its neighbors
* Variable assignments needs to be counted
  + CSP can be a class with a counter for this inside it
* Need a method for evaluating the AllDiff constraints on the 9 variables
  + For each variable, there will be 3 AllDiff evaluations – row, column and diagonal
    - So we need a method that returns the participating variables in the AllDiff constraint
* We need to pass a copy of the Sudoku to each recursive call, so that while backtracking we don’t have issues
* We need a converter for converting list of variables into a Sudoku grid, and viceversa
  + Let there be a SudokuGrid class for that
* Do we need a constraint graph

**Version 1**

* Make the algorithm generic so that version2 and version3 can be easily plugged in to version 1