1. **CSP formulation for sudoku**

Variable = Each empty box in the Grid

Domain = {1, 2, 3, 4, 5, 6 ,7, 8, 9}

Constrain = All the numbers in each row, column, and region must be different

**Code Overview:**

1. For all box in the Grid:
2. Find an un-assigned box
3. Assign a value for that box (in numeric order, 1 until 9) that do not violate the constrains
4. Repeat step(1) (recursion)
5. If we can’t assign any value, go back one box before (back tracking)
6. Print the Grid
7. **CSP formulation for nonogram:**

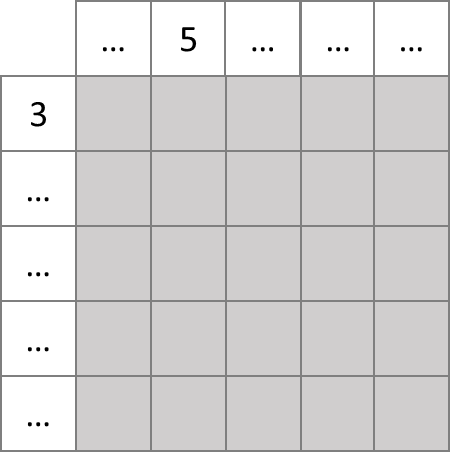
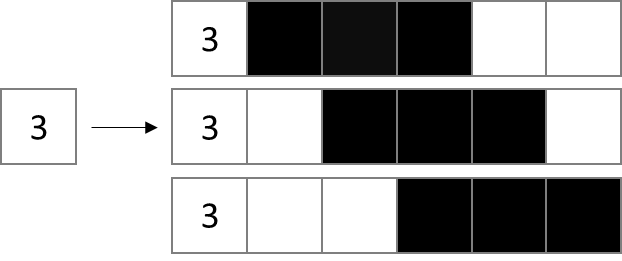
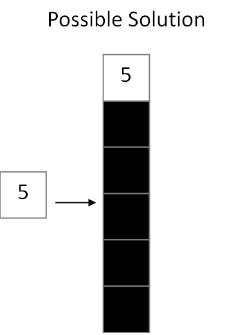
Variable = Grid Gi,j, i = number of row, j = number of column

Domain = {-1,0,1}\*, 1 = black, -1 = white, 0 = unknown (grey)

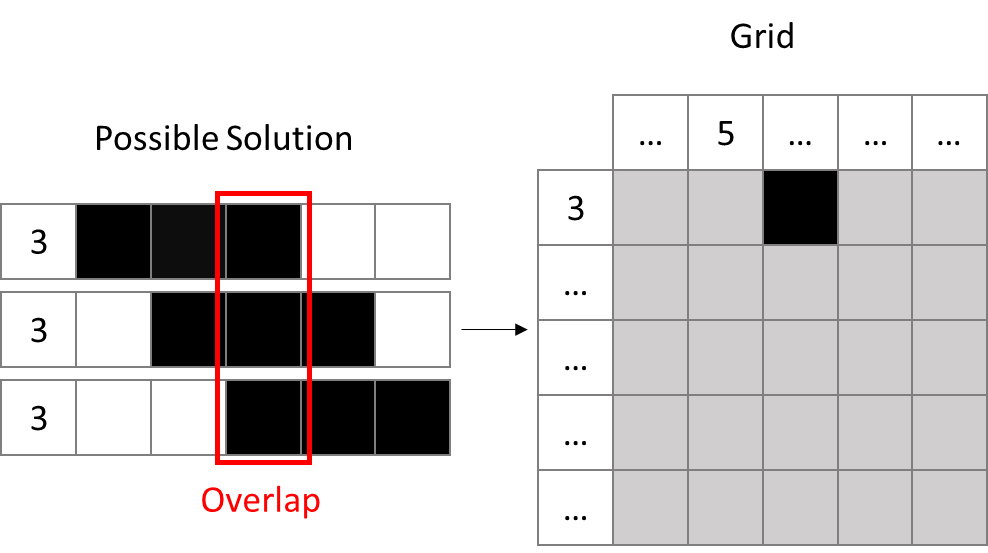
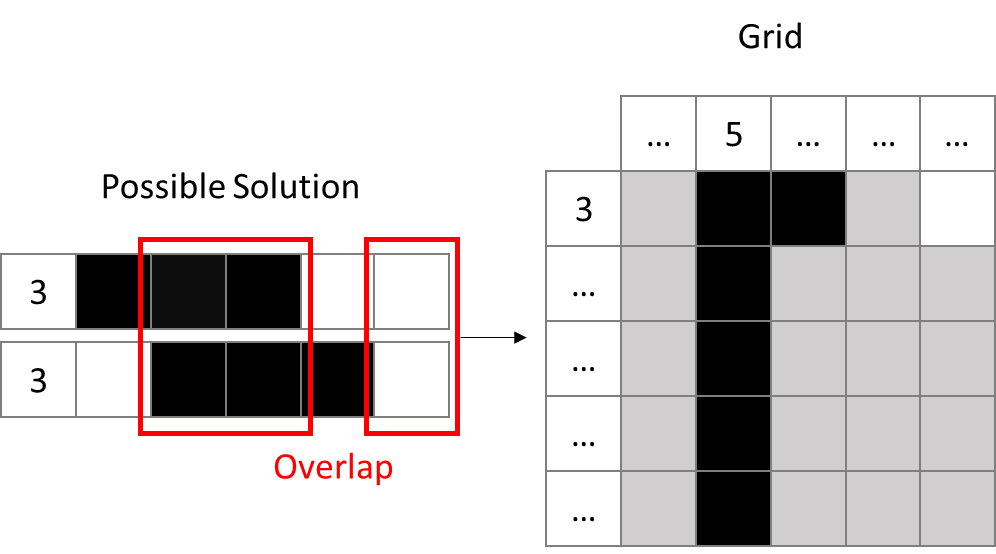
\*also, in the code you will see the domain as {“.”, “-”, “\*”}, “\*” = black, “.” = white, “-” = unknown

Constraint = The nonogram rule

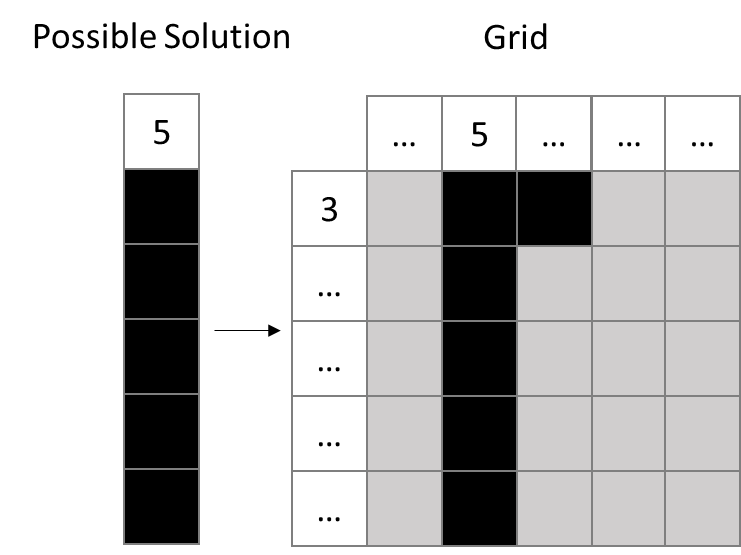
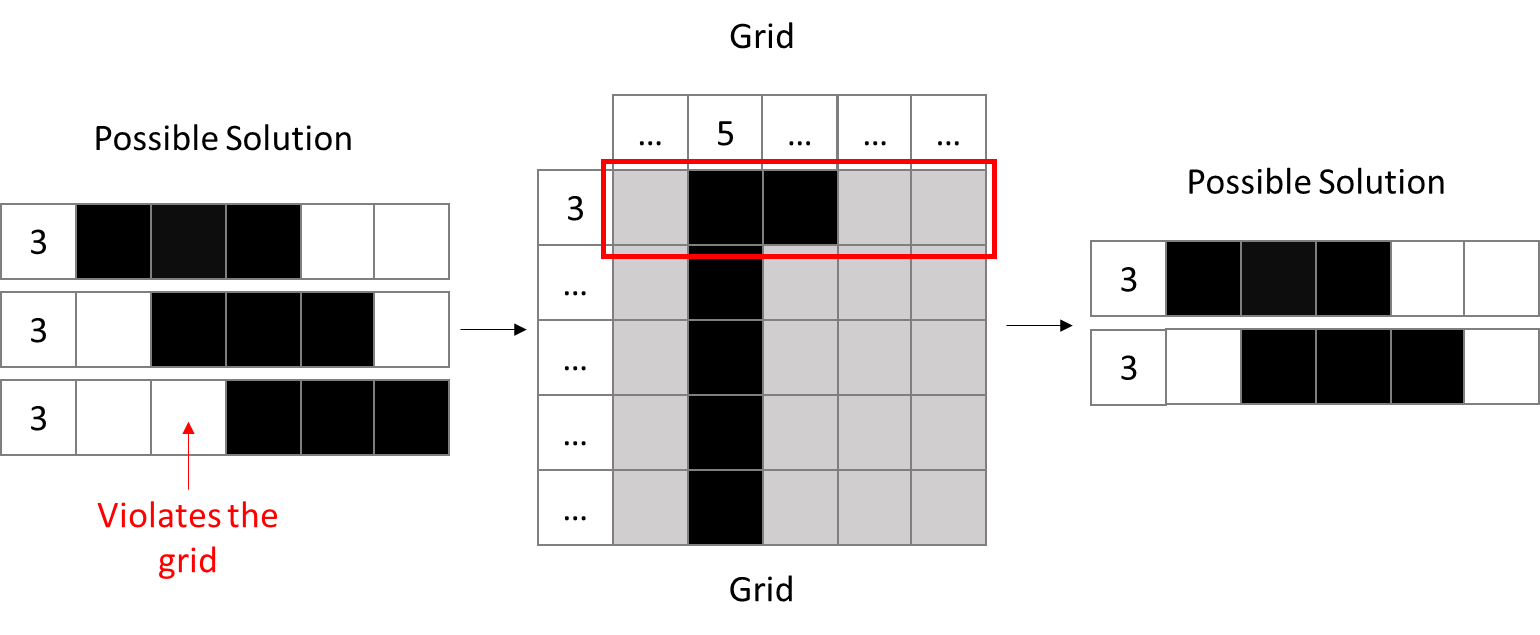
**Code overview:**

1. Get the nonogram size, rows configuration, and columns configuration from the terminal input
2. Create a Grid (Gi,j) with size = i x j, and initializes the Grid as grey (unknown)
3.  Generate all possible solutions for each row configuration and each column configuration

(Now, each row and each column have several possible solutions. The nonogram is complete if and only if every row and every column only have 1 possible solution)

1. While (number of possible solutions for every row > 1 or number of possible solutions for every column > 1):
2. For all possible solutions in each row, check overlap condition for black and white, if there is an overlap, then plot black or white on the Grid (Gi,j) respectively

(This part happens after step(4c))

1. Repeat step(4a) for each column
2. Compare all possible solutions in each row with the grid (Gi,j), if a possible solution violates the grid, then delete that solution
3. Repeat step(4c) for each column
4. Print the grid