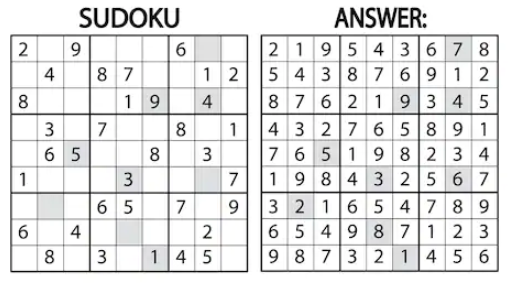
CIS 350/3501 Program 4

Due Dates: Part 1: August 9 Parts 1 & 2: August 16, 2019

Sudoku

[Sudoku](https://hackernoon.com/tagged/sudoku) is a number-placement [puzzle](https://hackernoon.com/tagged/puzzle) has the objective to fill a square grid of size n with numbers between 1 to n. The numbers are required to be placed so each row and column, and each of the square sub-grids (when present) contains all of the numbers from 1 to n.

The most common Sudoku puzzles use a 9 x 9 grid. The grids are partially filled, requiring a minimum of 17 numbers, so that the player can reach a solution.



In the answer, each row and column (9 x 9) has the values between 1 and 9, and each sub-grid (3 x 3) also contains the vlaues between 1 and 9.

Given a specific starting point of hints, there may be multiple correct answers to the puzzle.

A Sudoku solver can be created using a backtracking approach.

A candidate solution requirements:

* Each row and column has unique numbers – 1 to n – or empty spaces
* Each sub-graid has unique numbers – 1 to 9 – or empty spaces

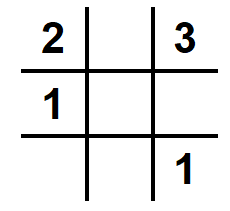
Alternate results:

* No empty locations to begin with
* No locations left and the candidate solution does not qualify as actual solution

Example of Approach to Solve (you may have create a different approach)

1. Create list of all the empty locations
2. Select location and place a number (between 1 and n) into that location
3. Validate the candidate grid
   1. If any requirement fails
      1. Abandon candidate solution
      2. Attempt next number
   2. If goal reached continue
4. Solution reached
   1. Yes – stop
   2. No – return to step 2 and repeat

Example (no sub-grids)



Empty locations (1,2) (2,2) (2,3) (3,1) (3,2)

Select (1,2), possible values 1, 2, 3

Place number 1 in location

Check if it meets requirements – it does

Select (2,2)

Place number 1 in location

Check if it meets requirements – fails two 1 in same row

Abandon candidate solution (backtrack from candidate solution)

Repeat with next number

Place number 2 in location

Check if it meets requirements – it does

Select (2,3)

Placing numbers 1, 2 and 3 in location fails requirements

Abandon solution, need to backtrack and change previous candidate solution “success”

Select (2,2)

Place number 3 in location

Check if requirements met – it does

Select (2,3)

Place number 2 in location

Check if requirements met – it does

Repeat until goal met or termination

Considerations:

* if starting hint (3,3) value is 2 rather than 1 – infeasible solution
* if hints (2,1) and (3,3) removed results in multiple solutions

A typical backtracking algorithm will use a recursive function for its implementation.

Therefore, the backtracking section of the program algorithm is simply:

Find row, column of an unassigned location

If no unassigned locations, return true

For digits from 1 to 9

a) If there is no requirement conflict for digit at row, col

assign digit to row, column and recursively try fill in rest of grid

b) If recursion successful, return true

c) Else, remove digit and try another

If all digits have been tried and nothing worked, return false

Potential methods:

selectUnassignedLocation

usedInRow

usedInColumn

usedInSubGrid

validToPlace (calls usedInRow, usedInColumn,usedInSubGrid)

printGrid

Now that there is a Sudoku Solver how do you create the grid – use the solver! Reuse!!!!!!

Creating the grid is based on 5 steps:

1. Generate a full grid of numbers (fully filled in). Cannot randomly generate numbers to fill in the grid. Need to make sure that the numbers are positioned on the grid following the Sudoku rules. To do so, use the sudoku solver algorithm applied to an empty grid. Need to add a random element to this solver algorithm to make sure that a new grid is generated every time it is run.
2. From the completed full grid, remove 1 value at a time.
3. For each value removed, run the sudoku solver algorithm to see if the grid can still be solved and to count the number of solutions it leads to.
4. If the resulting grid only has one solution, carry on the process from step 2. If not need to put the value taken away back in the grid.
5. Repeat the same process from step 2 several times using a different value each time to try to remove additional numbers, resulting in a more difficult grid to solve. The maximum number of removals results in 17 numbers in the partial grid.

Assignment:

You will work in a team of two to create the Sudoku Creater and Sudoku Solver. Note that there will be two program templates for this assignment.

The Soduko Solver algorithm needs to be able to run as part of the Sudoku Creator and independently. (Is there a way to create/use a common code base?)

The output of the Sudoku Creator program is the input to the Sudoku Solver program.

The programs will be based on a 9 x 9 grid with 3 x 3 sub-grids.

The output of the Sudoku Creator program is a file containing the partial grid in the form of:

row column value

Assumptions:

* Values entered will be integers
* The format of the input is correct but not the values.
  + That is, you still need to check that the values are between 1 and 9
* 0 will be treated as an invalid value
* User will enter input and output file names

That is, exception handling must be coded:

Input file doesn’t exist – error message to user, stop program

Empty input file – error message to user, stop program

Invalid data – error message to user, stop program

Program metrics:

You are to create metrics to see how efficient your program is – number of comparisons to solve/create grid, how many backtracking actions take place. You are to create three additional metrics.

Report:

You are to compare the metrics for multiple runs of your program – i.e. does the difficulty of the puzzle (ratio of hints) match the ratio of backtacking actions.

Screen and file output from the program:

User messages

Iterative output – when a sub-grid level is complete print out the grid as it looks at that point in time (e.g. when creating/solving grid – show grid after rows 1 through 3 have been completed)

Print final grid (partial for creator/solution for solver)