DAA Project on

Sudoku Solver

By

Sarthak Gupta **PES1201700077**

Dev Bhartra **PES1201700186**

B.Tech (CSE) Class 4G

PES University

Abstract:

Sudoku is a 9x9 matrix filled with numbers 1 to 9 in such a way that every row, column and sub-matrix (3x3) has each of the digits from 1 to 9. We are provided with a partially filled 9x9 matrix and have to fill every remaining cell in it.

The Naive Approach (Brute Force):

The Naive Algorithm is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found.

But this isn't the most efficient approach to this problem. Such an algorithm would have a runtime complexity of ***$O(N^{N^2})$***, where *N* is size of the Sudoku puzzle (i.e., 9 in classic Sudoku). The brute force algorithm would perform ***$2*10^{77}$* operations** to find a solution. That would not be practical.

Backtracking:

What is backtracking?

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

Backtracking applied to this problem:

Like all other Backtracking problems, we can solve the puzzle by assigning numbers sequentially to empty cells. Before assigning a number, we check whether the insertion of a certain number is 'valid' or not. The satisfaction of these condition define an insertion as valid. The same number should not be present in that row, column or in the smaller internal 3x3 sub-matrix.

After checking for validity, we assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, we then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, we return false.

Once there are no unallocated cells, the Sudoku is solved.

Worst case complexity for this is **$O(n^m)$**, where *n* is the number of possibilities for each square (i.e., 9 in classic Sudoku) and *m* is the number of spaces that are blank, which is rare. Actual tight bound is much much lesser than that, and very practical.

The Code:

https://github.com/devbhartra/SudokuSolver/blob/master/SOLVER.c (C code for 9x9)
https://github.com/devbhartra/SudokuSolver/blob/master/SOLVER.py (Python code for 2x2, 3x3, 4x4)

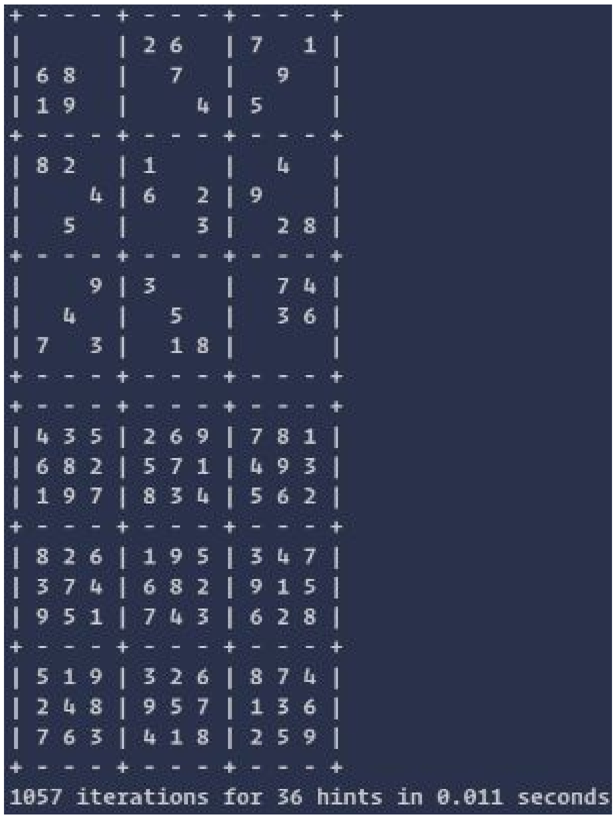
```c
#include <stdio.h>

int valid(int sudoku[9][9], int r, int c, int num);
int solve(int sudoku[9][9], int r, int c);
void print(int sudoku[9][9]);

int valid(int sudoku[9][9], int r, int c, int num) {
  int rStart = (r / 3) * 3, cStart = (c / 3) * 3;
  for (int i = 0; i < 9; i++) if (sudoku[r][i] == num || sudoku[i][c] == num
|| sudoku[rStart + (i % 3)][cStart + (i / 3)] == num) return 0;
  return 1;
}

int solve(int sudoku[9][9], int r, int c) {
  if (r > 9 && c > 9) return 1;
  if (sudoku[r][c]) {
    if (c + 1 < 9) return solve(sudoku, r, c + 1);
    else if (r + 1 < 9) return solve(sudoku, r + 1, 0);
    else return 1;
  } else {
    for (int i = 0; i < 9; i++) {
      if (valid(sudoku, r, c, i + 1)) {
        sudoku[r][c] = i + 1;
        if (solve(sudoku, r, c)) return 1;
        else sudoku[r][c] = 0;
      }
    }
  }
  return 0;
}

void print(int sudoku[9][9]) {
  printf("+ - - - + - - - + - - - +\n");
  for (int i = 0; i < 9; i++) {
    printf("|");
    for (int j = 0; j < 9; j++) {
      if (sudoku[i][j]) printf(" %d", sudoku[i][j]);
      else printf("  ");
      if (!((j + 1) % 3)) printf(" |");
    }
    printf("\n");
    if (!((i + 1) % 3)) printf("+ - - - + - - - + - - - +\n");
  }
}

int main(int argc, char const* argv[]) {
  int sudoku[9][9];
  if (argc < 2) { printf("Enter valid filename\n"); exit(1); }
  FILE* f = fopen(argv[1], "r");
  for (int i = 0; i < 9; i++) for (int j = 0; j < 9; j++) fscanf(f, "%d",
&sudoku[i][j]);
  fclose(f);
  print(sudoku);
  if (solve(sudoku, 0, 0)) print(sudoku);
  else printf("NO SOLUTION");
}
```

High Level Algorithm:

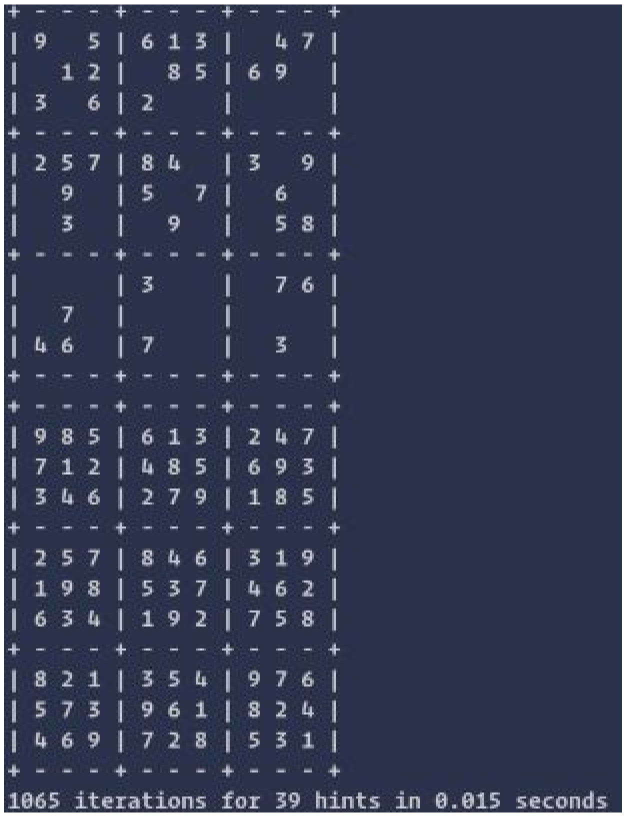
```
Algorithm SolveSudoku
     Input:  A 9x9 sudoku S, and a row position R & column position C
     Output: 1, with lexicographically first solution of the Sudoku in S, if a solution exists, else 0

if R > 9 and C > 9, return 1 // No cells left, sudoku is solved
if S(R, C) is filled, then // Try next cell
     if C <= 9, return SolveSudoku(S, R, C + 1) // Move to next column
     else if R <= 9, return SolveSudoku(S, R + 1, 0) // Row over, move to next row
     else return 1 // No cells left, sudoku is solved
else // R, C is yet to be filled
     for i ← 1 to 9, do: // Try every number
          if i is not already present in the same row, column, or sub-matrix as R, C in S, then // i is a valid entry
               fill i into S(R, C) // Try solving
               if SolveSudoku(S, R, C), return 1 // Solution is possible from here
               else reset S(R, C) to empty // Backtrack
return 0 // No solution found
```

Test Results:

The first figure shows the unsolved puzzle that is provided as input to the algorithm, and the figure below it is the solved output, followed by the number of iterations performed and given hints, along with the time taken for that particular run.



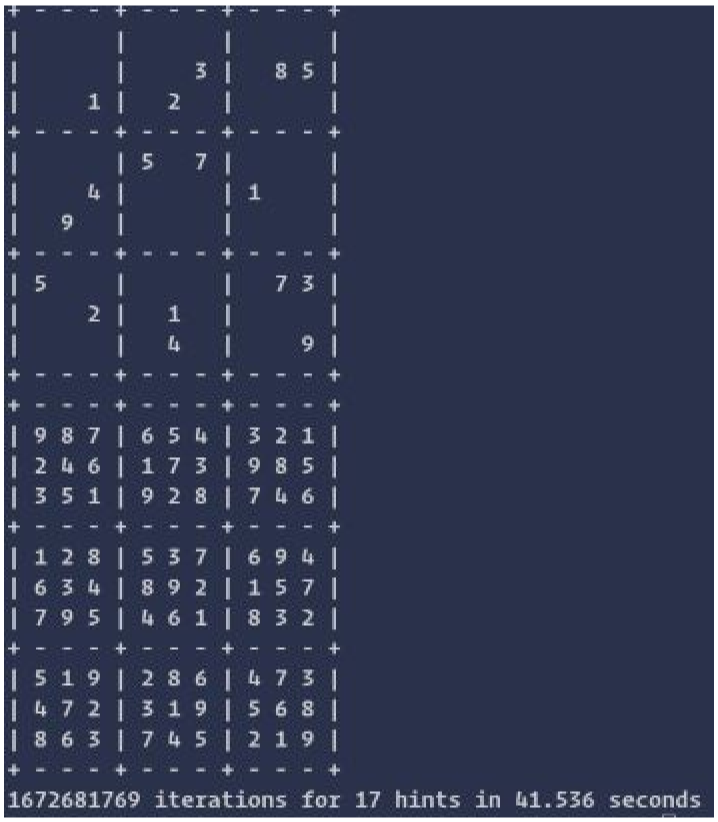
```
+ - - - + - - - + - - - +
|       | 2 6   | 7   1 |
| 6 8   |   7   |   9   |
| 1 9   |     4 | 5     |
+ - - - + - - - + - - - +
| 8 2   | 1     |   4   |
|     4 | 6   2 | 9     |
|   5   |     3 |   2 8 |
+ - - - + - - - + - - - +
|     9 | 3     |   7 4 |
|   4   |   5   |   3 6 |
| 7   3 |   1 8 |       |
+ - - - + - - - + - - - +
+ - - - + - - - + - - - +
| 4 3 5 | 2 6 9 | 7 8 1 |
| 6 8 2 | 5 7 1 | 4 9 3 |
| 1 9 7 | 8 3 4 | 5 6 2 |
+ - - - + - - - + - - - +
| 8 2 6 | 1 9 5 | 3 4 7 |
| 3 7 4 | 6 8 2 | 9 1 5 |
| 9 5 1 | 7 4 3 | 6 2 8 |
+ - - - + - - - + - - - +
| 5 1 9 | 3 2 6 | 8 7 4 |
| 2 4 8 | 9 5 7 | 1 3 6 |
| 7 6 3 | 4 1 8 | 2 5 9 |
+ - - - + - - - + - - - +
1057 iterations for 36 hints in 0.011 seconds
```



```
+ - - - + - - - + - - - +
| 9   5 | 6 1 3 |   4 7 |
|   1 2 |   8 5 | 6 9   |
| 3   6 | 2     |       |
+ - - - + - - - + - - - +
| 2 5 7 | 8 4   | 3   9 |
|   9   | 5   7 |   6   |
|   3   |   9   |   5 8 |
+ - - - + - - - + - - - +
|       | 3     |   7 6 |
|   7   |       |       |
| 4 6   | 7     |   3   |
+ - - - + - - - + - - - +
+ - - - + - - - + - - - +
| 9 8 5 | 6 1 3 | 2 4 7 |
| 7 1 2 | 4 8 5 | 6 9 3 |
| 3 4 6 | 2 7 9 | 1 8 5 |
+ - - - + - - - + - - - +
| 2 5 7 | 8 4 6 | 3 1 9 |
| 1 9 8 | 5 3 7 | 4 6 2 |
| 6 3 4 | 1 9 2 | 7 5 8 |
+ - - - + - - - + - - - +
| 8 2 1 | 3 5 4 | 9 7 6 |
| 5 7 3 | 9 6 1 | 8 2 4 |
| 4 6 9 | 7 2 8 | 5 3 1 |
+ - - - + - - - + - - - +
1065 iterations for 39 hints in 0.015 seconds
```

The following puzzle is has been designed to work against the brute force algorithm. It still takes 1,67,26,81,769 iterations with the help of backtracking.

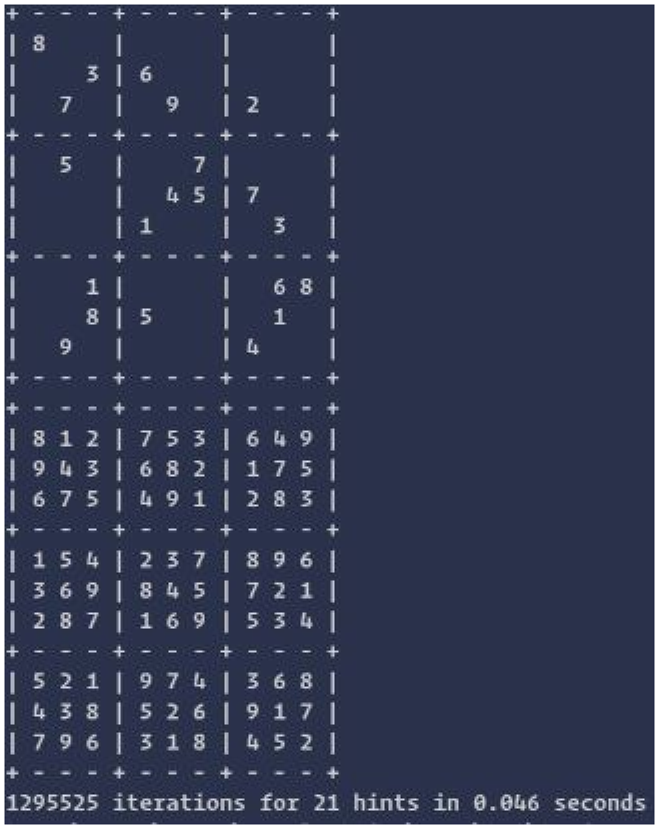
(https://en.wikipedia.org/wiki/Sudoku_solving_algorithms#/media/File:Sudoku_puzzle_hard_for_brute_force.svg)



```
+ - - - + - - - + - - - +
|       |       |       |
|       |     3 |   8 5 |
|     1 |   2   |       |
+ - - - + - - - + - - - +
|       | 5   7 |       |
|     4 |       | 1     |
|   9   |       |       |
+ - - - + - - - + - - - +
| 5     |       |   7 3 |
|     2 |   1   |       |
|       |   4   |     9 |
+ - - - + - - - + - - - +
+ - - - + - - - + - - - +
| 9 8 7 | 6 5 4 | 3 2 1 |
| 2 4 6 | 1 7 3 | 9 8 5 |
| 3 5 1 | 9 2 8 | 7 4 6 |
+ - - - + - - - + - - - +
| 1 2 8 | 5 3 7 | 6 9 4 |
| 6 3 4 | 8 9 2 | 1 5 7 |
| 7 9 5 | 4 6 1 | 8 3 2 |
+ - - - + - - - + - - - +
| 5 1 9 | 2 8 6 | 4 7 3 |
| 4 7 2 | 3 1 9 | 5 6 8 |
| 8 6 3 | 7 4 5 | 2 1 9 |
+ - - - + - - - + - - - +
1672681769 iterations for 17 hints in 41.536 seconds
```

The following Sudoku is considered by many to be one of the hardest to solve, and is specifically designed to be unsolvable to all but the sharpest minds:

(https://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html)

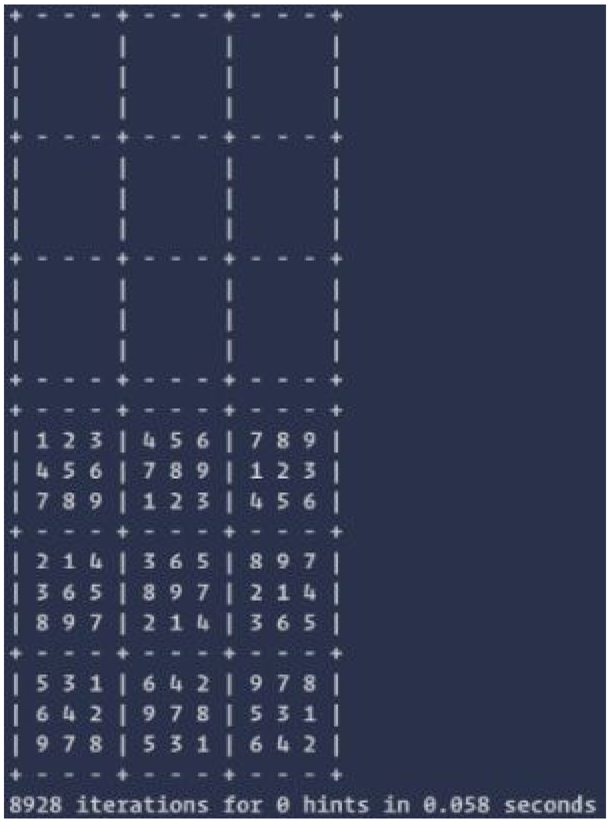


```
+ - - - + - - - + - - - +
| 8     |       |       |
|     3 | 6     |       |
|   7   |   9   | 2     |
+ - - - + - - - + - - - +
|   5   |     7 |       |
|       |   4 5 | 7     |
|       | 1     |   3   |
+ - - - + - - - + - - - +
|     1 |       |   6 8 |
|     8 | 5     |   1   |
|   9   |       | 4     |
+ - - - + - - - + - - - +
+ - - - + - - - + - - - +
| 8 1 2 | 7 5 3 | 6 4 9 |
| 9 4 3 | 6 8 2 | 1 7 5 |
| 6 7 5 | 4 9 1 | 2 8 3 |
+ - - - + - - - + - - - +
| 1 5 4 | 2 3 7 | 8 9 6 |
| 3 6 9 | 8 4 5 | 7 2 1 |
| 2 8 7 | 1 6 9 | 5 3 4 |
+ - - - + - - - + - - - +
| 5 2 1 | 9 7 4 | 3 6 8 |
| 4 3 8 | 5 2 6 | 9 1 7 |
| 7 9 6 | 3 1 8 | 4 5 2 |
+ - - - + - - - + - - - +
1295525 iterations for 21 hints in 0.046 seconds
```

The time taken here is 0.046 seconds, which is still a fraction of the 41.5 seconds taken to solve the previous test case.

Even when no hints are provided, the algorithm takes 0.058 seconds

This solution is lexicographically the first sudoku.



```
+ - - - + - - - + - - - +
|       |       |       |
|       |       |       |
|       |       |       |
+ - - - + - - - + - - - +
|       |       |       |
|       |       |       |
|       |       |       |
+ - - - + - - - + - - - +
|       |       |       |
|       |       |       |
|       |       |       |
+ - - - + - - - + - - - +
+ - - - + - - - + - - - +
| 1 2 3 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |
+ - - - + - - - + - - - +
| 2 1 4 | 3 6 5 | 8 9 7 |
| 3 6 5 | 8 9 7 | 2 1 4 |
| 8 9 7 | 2 1 4 | 3 6 5 |
+ - - - + - - - + - - - +
| 5 3 1 | 6 4 2 | 9 7 8 |
| 6 4 2 | 9 7 8 | 5 3 1 |
| 9 7 8 | 5 3 1 | 6 4 2 |
+ - - - + - - - + - - - +
8928 iterations for 0 hints in 0.058 seconds
```