Problem Solving Lab PROJECT
(20B16CS323)

THE SUDOKU SOLVER

Submitted by:

1) Ashutosh Dixit (19103059, B10)
2) Bhavesh Kumar Dixit (19103303, B10)
3) Utsav Dhankar (19104014, B11)

Submitted To:

Dr. Prashant Kaushik
Dr. K. Vimal Kumar

Department of CSE/IT
Jaypee Institute of Information Technology
University, Noida

YEAR 2022

1. Statement about the problem

Everyone who's everyone is crazy for this little logic puzzle that involving filling numbers into grid. The goal of Sudoku is to assign digits to the empty cells so that every row, column, and subgrid contains exactly one instance of the digits from 1 to 9. The starting cells are assigned to constrain the puzzle such that there is only one way to finish it. Sudoku solvers pride themselves on the fact that there is no need to "guess" to solve the puzzle, that careful application of logic will lead you to the solution. However, a computer solver can make and unmake guesses fast enough to not care, so let's just throw some recursive backtracking at it!
The objective is to fill a 9x9 grid with digits subject to the constraints that each column, each row, and each of the nine 3x3 sub-grids that compose the grid contains unique integers in [1,9]. The grid is initialized with partial assignment as show in the Fig(1); and a complete solution is shown in Fig(2).

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fig(1)

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Fig(2)

2. Reason/ Motivation to chose the topic

As we have recently learn practical implication of Data structures, we decided to go with this topic and then we decided to make something with real life application , so we decided to make this project.

3. Tools and Technologies used

- 2D Array

- Recursion
- Backtracking
- HTML, CSS, JavaScript

4. Dataset Description

Our dataset consists of a grid initialized with partial assignments in which an empty block is identified via '0' in int form.
We have used our dataset from this URL :
https://sugoku.herokuapp.com/board?difficulty=random
This generates a random 9x9 partially filled grid that is solvable.

Example Input :

```
[
 [0,3,0,9,1,0,0,0,0],      //row-1
 [0,4,0,0,6,0,0,0,0],      //row-2
 [0,0,0,0,0,0,0,0,0],      //row-3
 [0,0,2,4,0,0,0,0,0],      //row-4
 [0,0,0,0,0,0,2,3,0],      //row-5
 [7,0,0,6,2,0,4,0,5],      //row-6
 [0,0,0,0,0,6,0,0,0],      //row-7
 [8,0,0,0,0,1,0,4,0],      //row-8
 [0,7,0,8,0,2,5,6,0]       //row-9
 ]
```

5. Pseudocode of our strategy

In pseudocode, our strategy is:

Find row, col of an unassigned cell
If there is none, return true

For digits from 1 to 9
if there is no conflict for digit at row,col assign digit to row,col and recursively try fill in rest of grid
if recursion successful, return true
if !successful, remove digit and try another
if all digits have been tried and nothing worked, return false to trigger backtracking

6. Design of Our Project

The game is simply in HTML, CSS, and JavaScript. Taking about the features of this game, the PC controls of this game are also simple. First, you have a button "Generate New Puzzle" that will generate a new partially filled 9x9 Sudoku Board. You can use "Solve" button to solve the puzzle if you want.
This Hyperlink gives us a 9x9 grid which has integers in range [0,9] in which '0' denotes an empty cell.

There are mainly two functions that work when we click on "Get New Puzzle" or "Solve" button.
These two functions are:
1. function solveSudoku(board){

This function is a major function which uses recursion + bactracking to solve the puzzle (9x9 grid).

}

2. function changeBoard(board){

This function is a function that changes the board when it needs to updated i.e. whenever user requests for a new puzzle or whenever board gets completely Solved.

}

The HTML and CSS files uses the basic functionalities like color, font etc available to give interface and add style to our file.

7. Algorithm Implementation

A brute-force approach would be to try every possible assignment to empty entries, and then check if that assignment leads to a valid solution. This is wasteful, since if setting a value early on leads to a constant violation, there is no point in continuing. Therefore, we should apply the backtracking principle.
Specifically, we traverse the 2D array entries one at a time. If the entry is empty, we try each value for the entry, and see of the updated 2D array is still valid; if it is, we recurse. If all the entries have been filled, the search is successful. The naive approach to testing validity is wasteful. However, we can reduce runtime considerably by making use of the fact that we are adding a value to an array that already satisfies the constraints. This means that we need to check just the row, column and subgrid of the added entry.
For example, suppose we begin with lower-left entry for the configuration in Fig(1) on page 2. Adding 1 to entry does not violate any row, column, or subgrid constraint, so we move on to the next entry in that row. We cannot put a 1, since that would now violate a row constraint; however a 2 is acceptable.

This would be C++ implementation:

```
class Solution{public:


    bool rightToPlace(vector<vector<char>> board, int i, int j, char
num, int n)
    {
        // This is work only for 9 x 9 Sudoku problem.
        for (int x = 0; x < n; x++)
        {
```

```
            // This is check for ith row and jth column , If we found
element already then simply return false
            if (board[i][x] == num || board[x][j] == num)
            {
                return false;
            }
        }
        // This is for 3 x 3 sudoku part
        int x = (i / 3) * 3;
        int y = (j / 3) * 3;
        for (int p = x; p < x + 3; p++)
        {
            for (int q = y; q < y + 3; q++)
            {
                // If we find element here then simply return false
                if (board[p][q] == num)
                {
                    return false;
                }
            }
        }

        return true;
    }

    bool sudokuSolver(vector<vector<char>> &board, int i, int j, int
n)
    {
        //The first condition is i==n which occurs when we traverse
the entier matrix.
        //So if we can traverse the entire matrix then we found our
solution so return true to the parent.
        if (i == n)
        {
            return true;
        }
        // This condistion occurs when our currunt point is after
last element in row.
        // in short if a[i][9] which is out bound for matrix then we
need to go the next  row by i+1 and j=0
        // Here i am doing 0 base indexing so if you are in any other
language then formula might not work.

        // Mistake : make sure you write "return sudokuSolver(board,
```

```
i + 1, 0, n)" not just function call
        if (j == n)
        {
            return sudokuSolver(board, i + 1, 0, n);
        }
        // if element already had some value then there is no need to
change it, So basically move on for next element.
        if (board[i][j] != '.')
        {
            return sudokuSolver(board, i, j + 1, n);
        }
        // This is heart part of problem
        // here we are checking for 1 to 9, if element is match then
we go to next sub problem
        // else we backtrack to previos problem and change value or
parent (col - 1) element while col is not zero
        // if col become zero then pointer go to previous row . This
call backtracking.
        for (int z = 1; z <= 9; z++)
        {
            // rightToPlace fuction give true or false value
according to element sutable for this place or not.
            if (rightToPlace(board, i, j, '0' + z, n) == true)
            {

                board[i][j] = '0' + z;
                // Here we call subproblem and check for response. If
we got true it means subproblem will solve if we place element here.
                bool checkForNext = sudokuSolver(board, i, j + 1, n);
                if (checkForNext == true)
                {
                    return true;
                }
                // Mistake : If above case is not working then we
need to change currunt point value to "."
                // this is second heart step of backtracking :)
                board[i][j] = '.';
            }
        }
        // If not match any element from 1 to 9 then we return false
because in previos some where goes wrong!!
        return false;
    }
```

```
    // Main Sudoku Solver Fuction
    void solveSudoku(vector<vector<char>> &board)
    {
        // Here just pass value for sudokuSolver function...
        // board.size() must be 9
        bool x = sudokuSolver(board, 0, 0, board.size());
    }
};
```

8. Source code

index.html

```
<!DOCTYPE html><html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.m
in.css"
        integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

    <link rel="stylesheet" href="style.css">
    <script defer src="script.js"></script></head>
<body>
    <nav class="navbar navbar-light " style="font-size: 25px; font-
```

```
family: sans-serif; background-color: whitesmoke;">


        The Sudoku Solver Project

    </nav>
    <br><br>
    <div id="container">

        <div class="lsb tsb" id="0">

        </div>

        <div class="tsb ldb" id="1">

        </div>

        <div class="tsb ldb" id="2">

        </div>

        <div class="tsb lsb" id="3">

        </div>
        <div class="tsb ldb" id="4">

        </div>

        <div class="tsb ldb" id="5">

        </div>
        <div class="tsb lsb" id="6">

        </div>

        <div class="tsb ldb" id="7">

        </div>
        <div class="tsb rsb ldb" id="8">

        </div>

        <div class="lsb tdb" id="9">
```

```
        </div>

        <div class="ldb tdb" id="10">

        </div>

        <div class="ldb tdb" id="11">

        </div>

        <div class="lsb tdb" id="12">

        </div>
        <div class="ldb tdb" id="13">

        </div>

        <div class="ldb tdb" id="14">

        </div>
        <div class="lsb tdb" id="15">

        </div>

        <div class="ldb tdb" id="16">

        </div>
        <div class="rsb ldb tdb" id="17">

        </div>

        <div class="lsb bsb tdb" id="18">

        </div>

        <div class="bsb ldb tdb" id="19">

        </div>

        <div class="bsb ldb tdb" id="20">

        </div>

        <div class="bsb lsb tdb" id="21">
```

```
        </div>
        <div class="bsb ldb tdb" id="22">

        </div>

        <div class="bsb ldb tdb" id="23">

        </div>
        <div class="bsb lsb tdb" id="24">

        </div>

        <div class="bsb ldb tdb" id="25">

        </div>
        <div class="bsb rsb ldb tdb" id="26">

        </div>

        <div class="lsb" id="27">

        </div>

        <div class="ldb" id="28">

        </div>

        <div class="ldb" id="29">

        </div>

        <div class="lsb" id="30">

        </div>
        <div class="ldb" id="31">

        </div>

        <div class="ldb" id="32">

        </div>
        <div class="lsb" id="33">
```

```
        </div>

        <div class="ldb" id="34">

        </div>
        <div class="rsb ldb" id="35">

        </div>

        <div class="lsb tdb" id="36">

        </div>

        <div class="ldb tdb" id="37">

        </div>

        <div class="ldb tdb" id="38">
        </div>

        <div class="lsb tdb" id="39">

        </div>
        <div class="ldb tdb" id="40">

        </div>

        <div class="ldb tdb" id="41">

        </div>
        <div class="lsb tdb" id="42">

        </div>

        <div class="ldb tdb" id="43">

        </div>
        <div class="rsb ldb tdb" id="44">

        </div>

        <div class="lsb bsb tdb" id="45">

        </div>
```

```
        <div class="bsb ldb tdb" id="46">

        </div>

        <div class="bsb ldb tdb" id="47">

        </div>

        <div class="bsb lsb tdb" id="48">

        </div>
        <div class="bsb ldb tdb" id="49">

        </div>

        <div class="bsb ldb tdb" id="50">

        </div>
        <div class="bsb lsb tdb" id="51">

        </div>

        <div class="bsb ldb tdb" id="52">

        </div>
        <div class="bsb rsb ldb tdb" id="53">

        </div>

        <div class="lsb" id="54">

        </div>

        <div class="ldb" id="55">

        </div>

        <div class="ldb" id="56">

        </div>

        <div class="lsb" id="57">
```

```
        </div>
        <div class="ldb" id="58">

        </div class="ldb">

        <div class="ldb" id="59">

        </div>
        <div class="lsb" id="60">

        </div>

        <div class="ldb" id="61">

        </div>
        <div class="rsb ldb" id="62">

        </div>

        <div class="lsb tdb" id="63">

        </div>

        <div class="ldb tdb" id="64">

        </div>

        <div class="ldb tdb" id="65">

        </div>

        <div class="lsb tdb" id="66">

        </div>
        <div class="ldb tdb" id="67">

        </div>

        <div class="ldb tdb" id="68">

        </div>
        <div class="lsb tdb" id="69">

        </div>
```

```
        <div class="ldb tdb" id="70">

        </div>
        <div class="rsb ldb tdb" id="71">

        </div>

        <div class="lsb bsb tdb" id="72">

        </div>

        <div class="bsb ldb tdb" id="73">

        </div>

        <div class="bsb ldb tdb" id="74">

        </div>

        <div class="bsb lsb tdb" id="75">

        </div>
        <div class="bsb ldb tdb" id="76">

        </div>

        <div class="bsb tdb ldb" id="77">

        </div>
        <div class="bsb lsb tdb" id="78">

        </div>

        <div class="bsb ldb tdb" id="79">

        </div>
        <div class="bsb rsb ldb tdb" id="80">

        </div>


    </div>
    <br>
```

```
    <div id="buttons">
        <button type="button" class="btn btn-danger" id="generate-
sudoku">Get New
            Puzzle</button>    
        <button type="button" class="btn btn-success"
id="solve">Solve</button>
    </div></body>
</html>
```

style.css

```
#container{
    height: auto;
    width: 540px;
    background-color: white;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-evenly;
    align-content: space-evenly ;
    margin: 0 auto;
}
#generate-sudoku{
    margin:auto;
    display:block;;
}

#solve{
    margin:auto;
    display:block;;
}
#container div{
    background-color: whitesmoke;
    height: 60px;
    width: 60px;
    box-sizing: border-box;
    font-family: sans-serif;
    text-align: center;
    vertical-align: middle;
    line-height: 60px;
    font-size: 30px;
    color: green;

}
#container div:hover{
```

```
    background-color: lightskyblue;
}
.lsb{
    border-left: black 2px solid;
}
.bsb{
    border-bottom: black 2px solid;
}
.rsb{
    border-right: black 2px solid;
}
.tsb{
    border-top: black 2px solid;
}

.ldb{
    border-left: black 0.6px dashed;
}
.bdb{
    border-bottom: black 0.6px dashed;
}
.rdb{
    border-right: black 0.6px dashed;
}
.tdb{
    border-top: black 0.6px dashed;
}
```

script.js

```
var arr = [[], [], [], [], [], [], [], [], []]var temp = [[], [],
[], [], [], [], [], [], []]
for (var i = 0; i < 9; i++) {
    for (var j = 0; j < 9; j++) {
        arr[i][j] = document.getElementById(i * 9 + j);

    }
}
function initializeTemp(temp) {

    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            temp[i][j] = false;
```

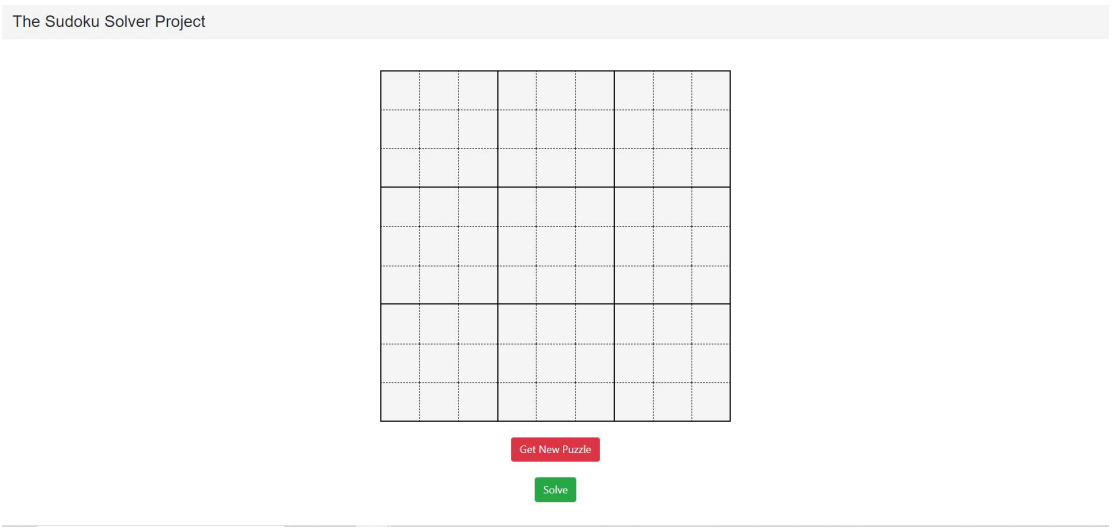
```
        }
    }
}

function setTemp(board, temp) {

    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            if (board[i][j] != 0) {
                temp[i][j] = true;
            }

        }
    }
}

function setColor(temp) {

    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            if (temp[i][j] == true) {
                arr[i][j].style.color = "#DC3545";
            }

        }
    }
}
function resetColor() {

    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {

            arr[i][j].style.color = "green";


        }
    }
}

var board = [[], [], [], [], [], [], [], [], []]

let button = document.getElementById('generate-sudoku')let solve =
document.getElementById('solve')
console.log(arr)function changeBoard(board) {
```

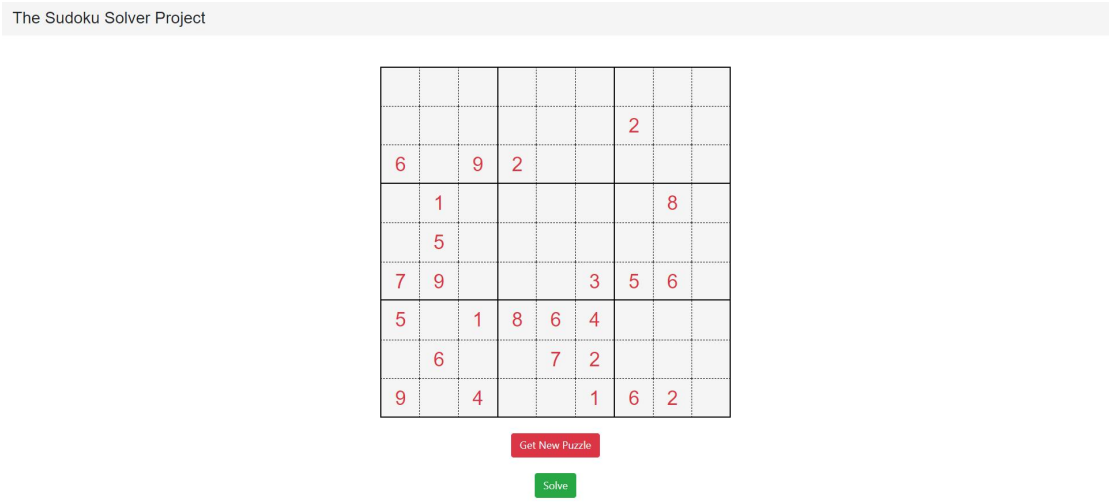
```
    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            if (board[i][j] != 0) {

                arr[i][j].innerText = board[i][j]
            }

            else
                arr[i][j].innerText = ''
        }
    }
}

button.onclick = function () {
    var xhrRequest = new XMLHttpRequest()
    xhrRequest.onload = function () {
        var response = JSON.parse(xhrRequest.response)
        console.log(response)
        initializeTemp(temp)
        resetColor()

        board = response.board
        setTemp(board, temp)
        setColor(temp)
        changeBoard(board)
    }
    xhrRequest.open('get',
'https://sugoku.herokuapp.com/board?difficulty=random')
    //we can change the difficulty of the puzzle the allowed values
of difficulty are easy, medium, hard and random
    xhrRequest.send()
}

//validity of a value checking function
function isPossible(board, sr, sc, val) {
    for (var row = 0; row < 9; row++) {
        if (board[row][sc] == val) {
            return false;
        }
    }

    for (var col = 0; col < 9; col++) {
        if (board[sr][col] == val) {
            return false;
```

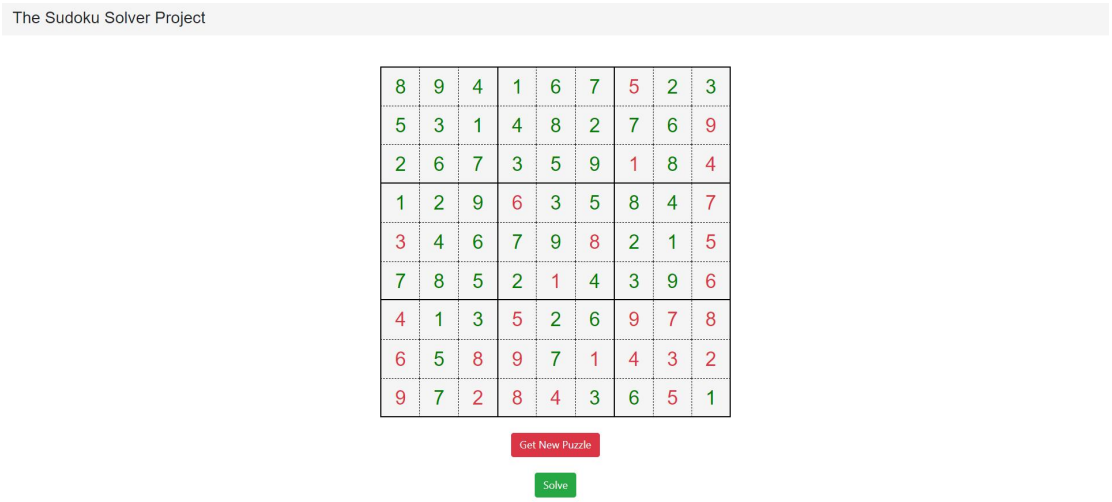
```
        }
    }

    var r = sr - sr % 3;
    var c = sc - sc % 3;

    for (var cr = r; cr < r + 3; cr++) {
        for (var cc = c; cc < c + 3; cc++) {
            if (board[cr][cc] == val) {
                return false;
            }
        }
    }
    return true;
}

//recursive helper function that tries values [1,9] for each empty
block
function solveSudokuHelper(board, sr, sc) {
    if (sr == 9) {
        changeBoard(board);
        return;
    }
    if (sc == 9) {
        solveSudokuHelper(board, sr + 1, 0)
        return;
    }
    //Non-empty block to be skipped
    if (board[sr][sc] != 0) {
        solveSudokuHelper(board, sr, sc + 1);
        return;
    }
    //putting values [1,9]
    for (var i = 1; i <= 9; i++) {
        //checking that value putten is valid
        if (isPossible(board, sr, sc, i)) {
            //if it's valid we will put that value and recurse to the
next block
            board[sr][sc] = i;
            solveSudokuHelper(board, sr, sc + 1);
            board[sr][sc] = 0;                          //backtracking
step
        }
    }
```

```
}

//main function taking partially filled board as a parameter
function solveSudoku(board) {
    solveSudokuHelper(board, 0, 0)
}

solve.onclick = function () {
    solveSudoku(board)
}
```

9. Preview







10. References

- Book : Elements Of Programming Interviews : The Insider's Guide
- Online Judge : Sudoku Solver - LeetCode and Valid Sudoku - LeetCode