# SUDOKU USING BACKTRACKING

Project submitted to the SRM University – AP, Andhra Pradesh
For the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology/Master of Technology**
In
**Computer Science and Engineering**
**School of Engineering and Sciences**

Submitted by
**CH.SANJEET**
**AP21110011402**

**SRM University–AP**
**Neerukonda, Mangalagiri, Guntur**
**Andhra Pradesh – 522 240**

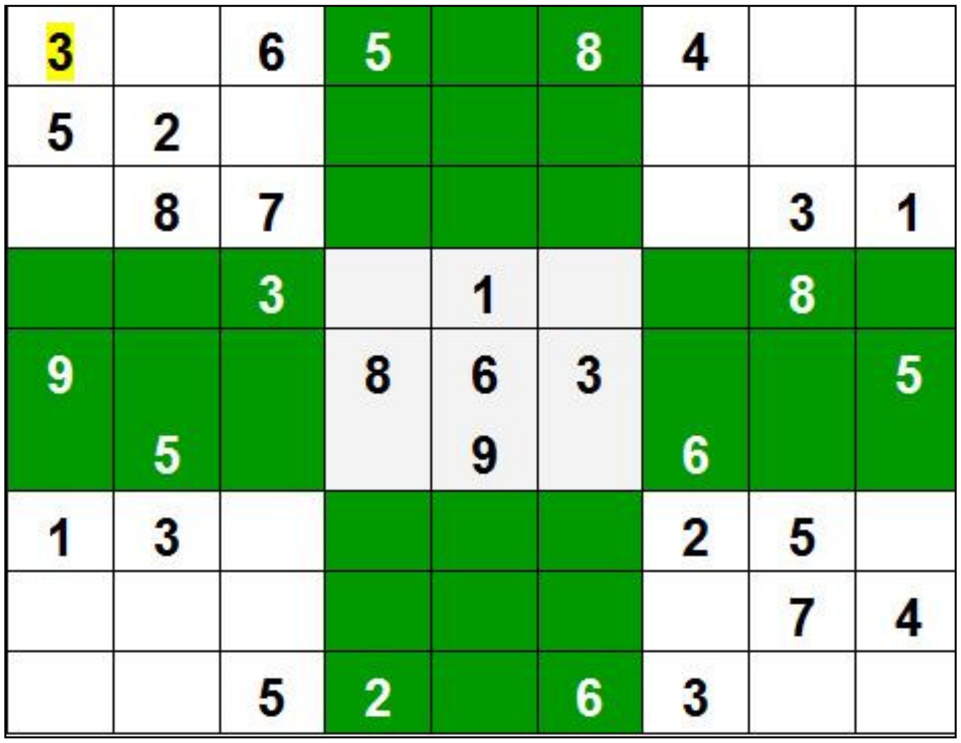
**Sudoku** is a logic based number placement puzzle.

Given a 9×9 grid, with some initial values filled. The goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and sub grid of size 3×3 contains exactly one instance of the digits from 1 to 9.

The objective is to fill all the empty cells of the grid with numbers such that it becomes valid.

A Sudoku is valid if it satisfies all the following properties:

1. Each row contains unique values ranging from 1 to 9.
2. Each column contains unique values ranging from 1 to 9.
3. Each of 9 sub-squares, of size 3×3, contains unique values from 1 to 9.



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

**Explanation/Algorithm:**

- Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.

- Create a recursive function that takes a grid.

- Check for any unassigned location.
    - If present then assigns a number from 1 to 9.
    - Check if assigning the number to current index makes the grid unsafe or not.
    - If safe then recursively call the function for all safe cases from 0 to 9.
    - If any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.

- If there is no unassigned location then return true.

## Source Code:

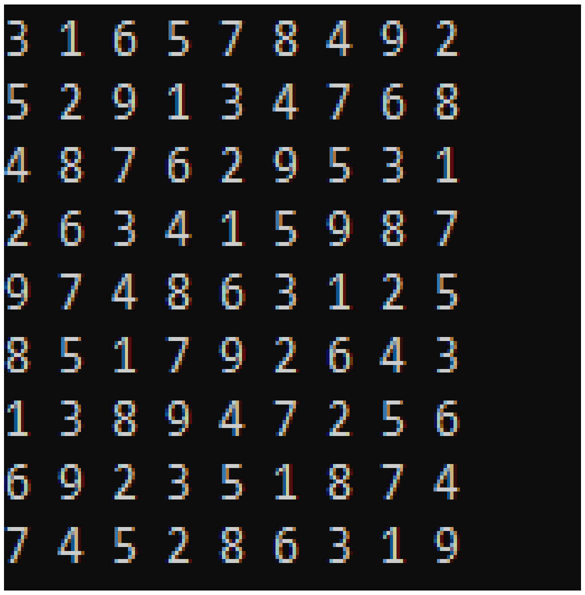
```
#include <bits/stdc++.h>

using namespace std;

#define UNASSIGNED 0

#define N 9

bool FindUnassignedLocation(int grid[N][N],int& row, int& col);

bool isSafe(int grid[N][N], int row,int col, int num);

bool SolveSudoku(int grid[N][N])

{

        int row, col;

        if (!FindUnassignedLocation(grid, row, col))

                return true;

        for (int num = 1; num <= 9; num++)

        {

                if (isSafe(grid, row, col, num))

                {

                        grid[row][col] = num;

                        if (SolveSudoku(grid))

                                return true;

                        grid[row][col] = UNASSIGNED;

                }

        }

        return false;

}
```

```
bool FindUnassignedLocation(int grid[N][N],int& row, int& col)
{
        for (row = 0; row < N; row++)
                for (col = 0; col < N; col++)
                        if (grid[row][col] == UNASSIGNED)
                                return true;
        return false;
}

bool UsedInRow(int grid[N][N], int row, int num)
{
        for (int col = 0; col < N; col++)
                if (grid[row][col] == num)
                        return true;
        return false;
}

bool UsedInCol(int grid[N][N], int col, int num)
{
        for (int row = 0; row < N; row++)
                if (grid[row][col] == num)
                        return true;
        return false;
}
```

```
bool UsedInBox(int grid[N][N], int boxStartRow,int boxStartCol, int num)
{
        for (int row = 0; row < 3; row++)
                for (int col = 0; col < 3; col++)
                        if (grid[row + boxStartRow][col + boxStartCol] ==num)
                                return true;
        return false;
}
bool isSafe(int grid[N][N], int row,int col, int num)
{
        return  !UsedInRow(grid, row, num)&& !UsedInCol(grid, col, num)
                && !UsedInBox(grid, row - row % 3,col - col % 3, num)
                && grid[row][col] == UNASSIGNED;
}

void printGrid(int grid[N][N])
{
        for (int row = 0; row < N; row++)
        {
                for (int col = 0; col < N; col++)
                        cout << grid[row][col] << " ";
                cout << endl;
        }
}
int main()
{
```

```
            int grid[N][N] =    {{ 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                                 { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                                 { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                                 { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                                 { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                                 { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                                 { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                                 { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                                 { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };
    if (SolveSudoku(grid) == true)
        printGrid(grid);
    else
        cout << "No solution exists";
    return 0;
}
```

**Output:**



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