



# **Project Report**

## **Sudoku Solver using Backtracking**

**Subject Name – Design and Analysis of Algorithms**

**Subject Code – 23CSH-301**

**Submitted To:**

**Er. Mohammad Shaqlain  
(E17211)**

**Submitted By:**

**Name:Abhi Kumar  
UID: 23BCS12097  
Section: KRG\_2-B**

*An implementation of recursive backtracking algorithm to solve 9×9 Sudoku puzzles*

## 1. Aim

To develop and analyze the complexity of a program to solve a Sudoku puzzle using the Backtracking algorithm.

## 2. Objective

The objective is to implement a Sudoku Solver that fills a  $9 \times 9$  grid by assigning digits (1–9) such that:

- Each row, column, and  $3 \times 3$  subgrid contains all digits exactly once.
- The program efficiently backtracks to correct conflicts.

## 3. Input / Apparatus Used

- **Programming Language:** C++
- **IDE Used:** Code::Blocks / VS Code / Visual Studio
- **Input:**  $9 \times 9$  grid with 0s as empty cells

## 4. Procedure / Algorithm

### Algorithm: Sudoku Solver (Backtracking)

1. Start
2. Find an empty cell (value = 0)
3. Try digits 1 to 9:
  - Check if safe in row, column, and  $3 \times 3$  subgrid
4. If safe, place and recurse
5. If conflict later, backtrack (remove and try next)
6. Repeat until board is full or no solution
7. Stop

### Recursive Equation:

Let  $n$  = current cell index

$$SolveSudoku(n) = \begin{cases} SolveSudoku(n+1) & \text{if valid placement} \\ \text{backtrack \& try next otherwise} \end{cases}$$

## 5. C++ Program

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define N 9
5
6 bool isSafe(int grid[N][N], int row, int col, int num) {
7     for (int x = 0; x < 9; x++)
8         if (grid[row][x] == num || grid[x][col] == num)
9             return false;
10
11    int startRow = row - row % 3;
12    int startCol = col - col % 3;
13    for (int i = 0; i < 3; i++)
14        for (int j = 0; j < 3; j++)
15            if (grid[i + startRow][j + startCol] == num)
16                return false;
17
18    return true;
19}
20
21 bool solveSudoku(int grid[N][N], int row, int col) {
22     if (row == N - 1 && col == N)
23         return true;
24
25     if (col == N) {
26         row++;
27         col = 0;
28     }
29
30     if (grid[row][col] != 0)
31         return solveSudoku(grid, row, col + 1);
32
33     for (int num = 1; num <= 9; num++)
34         if (isSafe(grid, row, col, num))
35         {
36             grid[row][col] = num;
37             if (solveSudoku(grid, row, col + 1))
38                 return true;
39             grid[row][col] = 0;
40         }
41     }
42 }
43
44 void printGrid(int grid[N][N]) {
45     for (int r = 0; r < N; r++) {
46         for (int d = 0; d < N; d++) {
47             cout << grid[r][d] << " ";
48         }
49         cout << endl;
```

```
50     }
51 }
52
53 int main() {
54     int grid[N][N] = {
55         {3, 0, 6, 5, 0, 8, 4, 0, 0},
56         {5, 2, 0, 0, 0, 0, 0, 0, 0},
57         {0, 8, 7, 0, 0, 0, 0, 3, 1},
58         {0, 0, 3, 0, 1, 0, 0, 8, 0},
59         {9, 0, 0, 8, 6, 3, 0, 0, 5},
60         {0, 5, 0, 0, 9, 0, 6, 0, 0},
61         {1, 3, 0, 0, 0, 0, 2, 5, 0},
62         {0, 0, 0, 0, 0, 0, 0, 7, 4},
63         {0, 0, 5, 2, 0, 6, 3, 0, 0}
64     };
65
66     if (solveSudoku(grid, 0, 0))
67         printGrid(grid);
68     else
69         cout << "No solution exists";
70     return 0;
71 }
```

Listing 1: Sudoku Solver in C++

### Sample 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
```

## 6. Complexity Analysis

- **Worst Case Time:**  $O(9^{N \times N})$  (empty grid)
- **Average Case:** Much faster due to pruning
- **Space Complexity:**  $O(N \times N)$  (grid + recursion stack)

## 7. Result

The program successfully solves the given Sudoku puzzle using backtracking. It efficiently fills all empty cells while satisfying all constraints.

## 8. Conclusion

- Demonstrates recursive backtracking for constraint satisfaction problems.
- Intelligent pruning reduces exponential search space.
- Strengthens understanding of algorithm design and recursion in C++.