**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

JNANA SANGAMA, BELAGAVI – 590 018

****

**A Mini Project Report on**

**SUKODU**

**Submitted by**

**Candidate Name**

|  |  |
| --- | --- |
| **NAME** | **USN** |
| **Siddharth Kesarwani** | **1RN20IS159** |
| **Sidharth S Pai** | **1RN20IS160** |
| **Surabhi S** | **1RN20IS168** |
| **Vipul Gaurav** | **1RN201S184** |

**Under the Guidance of**

|  |
| --- |
| **Faculty In charge**  **Dr. Suresh L**  **Designation: Professor and HOD**  **Dept. of ISE, RNSIT** |

**Department of Information Science and Engineering**

**RNS Institute of Technology**

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,

Bengaluru – 560 098

2021 – 2022

TABLE OF CONTENTS

[ABSTRACT 3](#_Toc112623371)

[INTRODUCTION 4](#_Toc112623372)

[1.1 Introduction 4](#_Toc112623373)

[1.2 Objectives 5](#_Toc112623374)

[1.3 Key Features 5](#_Toc112623375)

[ALGORITHM 6](#_Toc112623376)

[2.1 Algorithm Design Technique Used 6](#_Toc112623377)

[2.2 Algorithm 6](#_Toc112623379)

[DESIGN 7](#_Toc112623380)

[3.1 Flowchart 7](#_Toc112623381)

[SYSTEM REQUIREMENTS 8](#_Toc112623382)

[IMPLEMENTATION MODULES 9](#_Toc112623383)

[CODE 10](#_Toc112623384)

[RESULT 14](#_Toc112623385)

[TIME COMPLEXITY 15](#_Toc112623386)

[APPLICATIONS 16](#_Toc112623387)

[CONCLUSION & FURTHER ENHANCEMENTS 17](#_Toc112623388)

[10.1 Conclusion 17](#_Toc112623389)

[10.2 Further Enhancements 17](#_Toc112623390)

[10.3 References 17](#_Toc112623394)

# ABSTRACT

Sudoku is a logicbased number placement puzzle. The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes (also called blocks or regions) contains the digits from 1 to 9, only one time each (that is, exclusively). The puzzle setter provides a partially completed grid. Many computational methods to solve the puzzle like Sudoku had been developed in many ways. In this paper I am trying to create a Sudoku solving method by using the Empirical modelling Method based on the knowledge from the tuition and the pioneers.

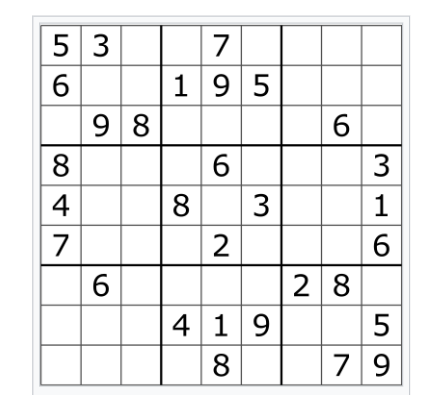
**CHAPTER 1:**

# INTRODUCTION

## 1.1 Introduction

**Sudoku** (originally called **Number Place**) is a logic -based, combinatorial number-placement puzzle. In classic Sudoku, the objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

French newspapers featured variations of the Sudoku puzzles in the 19th century, and the puzzle has appeared since 1979 in puzzle books under the name Number Place.[[5]](https://en.wikipedia.org/wiki/Sudoku#cite_note-Smith-5) However, the modern Sudoku only began to gain widespread popularity in 1986 when it was published by the Japanese puzzle company Nikoli under the name Sudoku, meaning "single number". It first appeared in a U.S. newspaper, and then *The times* (London), in 2004, thanks to the efforts of Wayne Gould, who devised a computer program to rapidly produce unique puzzles.



A typical Sudoku puzzle

## 1.2 Objectives

Sudoku is a logicbased number placement puzzle. The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes (also called blocks or regions) contains the digits from 1 to 9, only one time each.

## 1.3 Key Features

* Sudoku is played on a grid of 9 x 9 spaces.
* Within the rows and columns are 9 “squares” (made up of 3 x 3 spaces).
* Each row, column and square (9 spaces each) needs to be filled out with the numbers 1-9, without repeating any numbers within the row, column or square.

**CHAPTER 2:**

# ALGORITHM

## 2.1 Algorithm Design Technique Used

## BACKTRACKING is a technique based on algorithm to solve SUDOKU problem. It uses recursive calling to find the solution by building a solution step by step increasing values with time. It removes the solutions that doesn't give rise to the solution of the problem based on the constraints given to solve the problem.

## 2.2 Algorithm

1. 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.

2. Create a recursive function that takes a grid.

3. Check for any unassigned location. If present then assign 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.

4. If there is no unassigned location then return true.

**Sudoku** (grid)

find unfilled cell (i,j) in grid  
if (all the cells are filled) then  
   A valid sudoku is obtained hence return true

for each num in 1 to 9  
   if (cell (i,j) can be filled with num) then

fill it with num temporarily to check  
   if (**sudoku**(grid) is true) then

return true  
   if (the cell (i,j) can't be filled with num) then

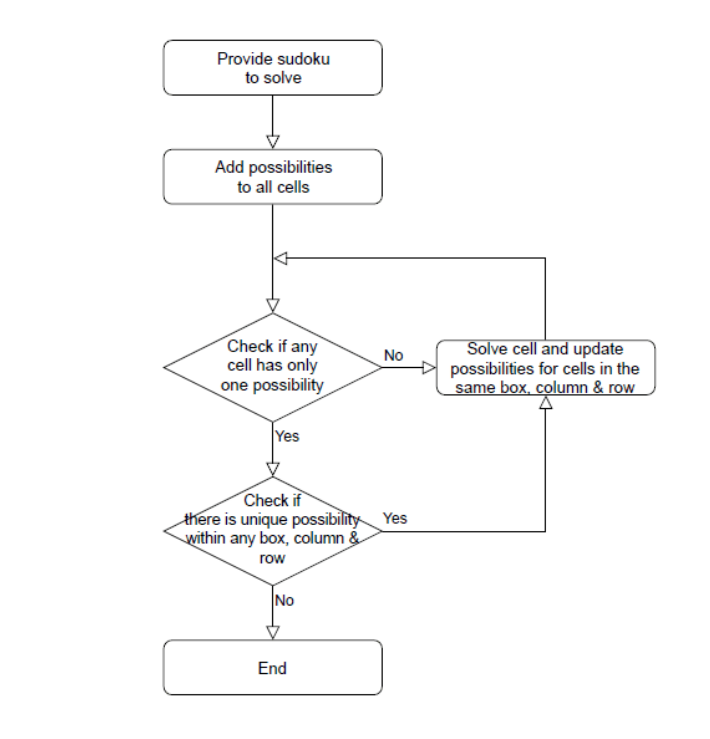
mark it as unfilled to trigger backtracking  
 if (none of the numbers from 1 to 9 can be filled in cell (i,j)) then

return false as there is no solution for this sudoku

**CHAPTER 3:**

# DESIGN

## 3.1 Flowchart



**CHAPTER 4:**

# SYSTEM REQUIREMENTS

**4.1 Hardware Requirements**

• **Processor:** Intel Core2 Quad @ 2.4Ghz on Windows® Vista 64-Bit / Windows® 7 64-Bit / Windows® 8 64-Bit / Windows® 8.1 64-Bit.

• **RAM:** 2GB of RAM

• **Memory:** 256GB Hard drive

• **Keyboard:** MS compatible keyboard

• **Mouse:** MS compatible mouse

**4.2 Software Requirements**

• **Operating system:** Windows® Vista 64-Bit / Windows® 7 64-Bit / Windows® 8 64-Bit / Windows® 8.1 64-Bit.

• **Front end Programming language:** Java

• **IDE:** Eclipse

**CHAPTER 5:**

# IMPLEMENTATION MODULES

* public class GFG
* public static Boolean solveSudoku
* public static void print(board [r][d])

**CHAPTER 6:**

# CODE

package sudoku;

public class GFG

{

public static boolean isSafe(int[][] board,

int row, int col,

int num)

{

for (int d = 0; d < board.length; d++)

{

if (board[row][d] == num) {

return false;

}

}

for (int r = 0; r < board.length; r++)

{

if (board[r][col] == num)

{

return false;

}

}

int sqrt = (int)Math.sqrt(board.length);

int boxRowStart = row - row % sqrt;

int boxColStart = col - col % sqrt;

for (int r = boxRowStart;

r < boxRowStart + sqrt; r++)

{

for (int d = boxColStart;

d < boxColStart + sqrt; d++)

{

if (board[r][d] == num)

{

return false;

}

}

}

return true;

}

public static boolean solveSudoku(

int[][] board, int n)

{

int row = -1;

int col = -1;

boolean isEmpty = true;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (board[i][j] == 0)

{

row = i;

col = j;

isEmpty = false;

break;

}

}

if (!isEmpty) {

break;

}

}

if (isEmpty)

{

return true;

}

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

{

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

{

board[row][col] = num;

if (solveSudoku(board, n))

{

return true;

}

else

{

board[row][col] = 0;

}

}

}

return false;

}

public static void print(

int[][] board, int N)

{

for (int r = 0; r < N; r++)

{

for (int d = 0; d < N; d++)

{

System.out.print(board[r][d]);

System.out.print(" ");

}

System.out.print("\n");

if ((r + 1) % (int)Math.sqrt(N) == 0)

{

System.out.print("");

}

}

}

public static void main(String args[])

{

int[][] board = new int[][] {

{ 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 }

};

int N = board.length;

if (solveSudoku(board, N))

{

print(board, N);

}

else {

System.out.println("No solution");

}

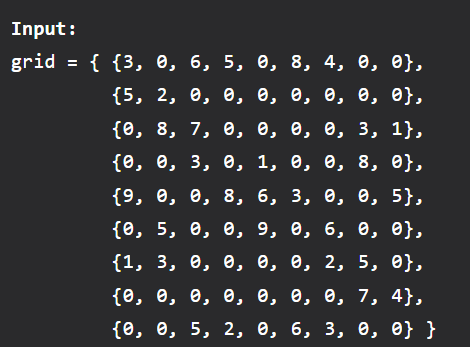
}

}

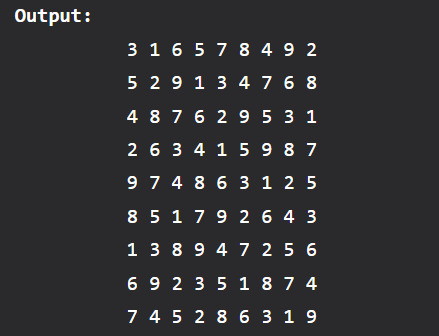
**CHAPTER 7:**

# RESULT

* Input



* Output



**CHAPTER 8:**

# TIME COMPLEXITY

Time complexity: O(9^(n\*n)).

For every unassigned index, there are 9 possible options so the time complexity is O(9^(n\*n)). The time complexity remains the same but there will be some early pruning so the time taken will be much less than the naive algorithm but the upper bound time complexity remains the same.

Space Complexity: O(n\*n).

To store the output array a matrix is needed.

**CHAPTER 9:**

# APPLICATIONS

* To Find All Hamiltonian Paths Present in a Graph.
* To Solve the N Queen Problem.
* Maze Solving Problems.
* The Knight's Tour Problem.

**CHAPTER 10:**

# CONCLUSION & FURTHER ENHANCEMENTS

## 10.1 Conclusion

## 10.2 Further Enhancements

## These are some of the main suggestions given by the user for improvement.

## Instant help by giving hints and users progress while playing.

## Score system based on time and accuracy and database to keep track of ten top record.

## 10.3 References

## <https://www.geeksforgeeks.org>

## [https://www.codechef.com](https://www.codechef.com/)

## [https://www.techgig.com](https://www.techgig.com/)

## <https://www.google.com/maps>