**A**

**Project Report**

On

**SUDOKU GAME SOLVER**

(CSE III Semester Mini project)

2019-2023

****

**Submitted To: Submitted By:**

Mrs. Meenakshi Maindola Sona verma

Prof. of Department (CSE) Roll no: 1918733

CSE-F-III-Sem

Session:2019-23

**Department of Computer Science & Engineering**

**Acknowledgement**

I am profoundly grateful to Mrs. Meenakshi Maindola for her expert guidance and continuous encouragement throughout to see that this project rights its target.

At last I must express my sincere heartfelt gratitude to all the staff members of Computer Engineering Department who helped us directly or indirectly during this course of work.

Sona Verma

**ABSTRACT**

In the last decade, solving the Sudoku puzzle has become every one’s passion. The simplicity of puzzle’s structure and the low requirement of mathematical skills caused people to have enormous interest in accepting challenges to solve the puzzle. Therefore, developers have tried to find algorithms in order to generate the variety of puzzles for human players so that they could be even solved by computer programming. In this essay, we have presented an algorithm called pencil-and-paper using human strategies. The purpose is to implement a more efficient algorithm and then compare it with another Sudoku solver named as brute force algorithm. This algorithm is a general algorithm that can be employed in to any problems. The results have proved that the pencil-and-paper algorithm solves the puzzle faster and more effective than the brute force algorithm

**Introduction**

Currently, Sudoku puzzles are becoming increasingly popular among the people all over the world. The game has become popular now in a large number of countries and many developers have tried to generate even more complicated and more interesting puzzles. Today, the game appears in almost every newspaper, in books and in many websites.

Sudoku is a logic puzzle. You can try playing Sudoku yourself here: Play Sudoku Online. In this project, you are going to implement a C++ program which solves Sudoku puzzles for you. Below is brief description of Sudoku and its rules:

Sudoku is a logic puzzle. The objective is to fill a 9×9 grid with digits in such a way such that each column, each row, and each of the nine 3×3 grids that make up the larger 9×9 grid contains all of the digits from 1 to 9 exactly once.

Each Sudoku puzzle begins with some cells filled in. These numbers are chosen such that there is a unique solution to the Sudoku.

**Constraints:**

• Sudoku puzzle can only contain the numbers 1 through 9.

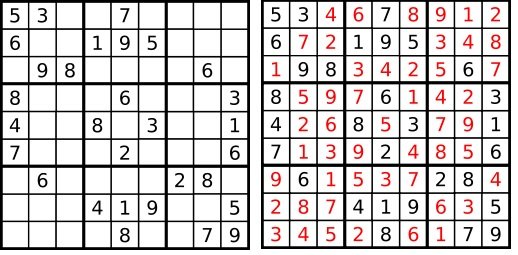
• A position constraint: Only 1 number can occupy a cell

• A row constraint: Only 1 instance of a number can be in the row

• A column constraint: Only 1 instance of a number can be in a column

• A region constraint: Only 1 instance of a number can be in a region

A typical Sudoku puzzle and its solution



Since you are going to be writing a C++ program to solve Sudoku, here's what the sample interaction would look like:

**Pseudo Code**

#include <iostream>

using namespace std;

// N is the size of the 2D matrix N\*N

#define N 9

/\* A utility function to print grid \*/

void print(int arr[N][N])

{

cout<<"SOLUTION FOR THE SUDOKU...."<<endl<<endl;

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

{

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

cout << arr[i][j] << " ";

cout << endl;

}

}

// Checks whether it will be legal to assign num to the given row, col

bool isSafe(int grid[N][N], int row,

int col, int num)

{

// Check if we find the same num in the similar row , we return false

for (int x = 0; x <= 8; x++)

if (grid[row][x] == num)

return false;

// Check if we find the same num in the similar column , we return false

for (int x = 0; x <= 8; x++)

if (grid[x][col] == num)

return false;

// Check if we find the same num in the particular 3\*3 matrix,we return false

int startRow = row - row % 3,

startCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (grid[i + startRow][j +startCol] == num)

return false;

return true;

}

/\* Takes a partially filled-in grid and attempts to assign values to all unassigned locations in such a way to meet the requirements for Sudoku solution (non-duplication across rows, columns, and boxes) \*/

bool solveSuduko(int grid[N][N], int row, int col)

{

// Check if we have reached the 8th row and 9th column (0 indexed matrix) , we are returning true to avoid further backtracking

if (row == N - 1 && col == N)

return true;

// Check if column value becomes 9 , we move to next row and column start from 0

if (col == N) {

row++;

col = 0;

}

// Check if the current position of the grid already contains value >0, we iterate for next column

if (grid[row][col] > 0)

return solveSuduko(grid, row, col + 1);

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

{

// Check if it is safe to place the num (1-9) in the given row ,col ->we move to next column

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

{

/\* Assigning the num in the current (row,col) position of the grid and assuming our assigned num in the position is correct \*/

grid[row][col] = num;

// Checking for next possibility with next column

if (solveSuduko(grid, row, col + 1))

return true;

}

/\* Removing the assigned num , since our assumption was wrong , and we go for next assumption with diff num value\*/

grid[row][col] = 0;

}

return false;

}

// Driver Code

int main()

{

// 0 means unassigned cells

int grid[N][N];

cout<<"enter 9 x 9 Grid(Use 0 for empty)"<<endl;

for(int i=0;i<9;i++){

for(int j=0;j<9;j++){

cout<<"Enter values for["<<i<<"]["<<j<<"]"<<" : ";

cin>>grid[i][j];

}

}

cout<<endl<<"YOU HAVE ENTERED"<<endl;

for(int i=0;i<9;i++){

for(int j=0;j<9;j++){

cout<<"\t"<<grid[i][j];

}

cout<<endl;

}

cout<<endl;

cout<<endl;

if (solveSuduko(grid, 0, 0))

print(grid);

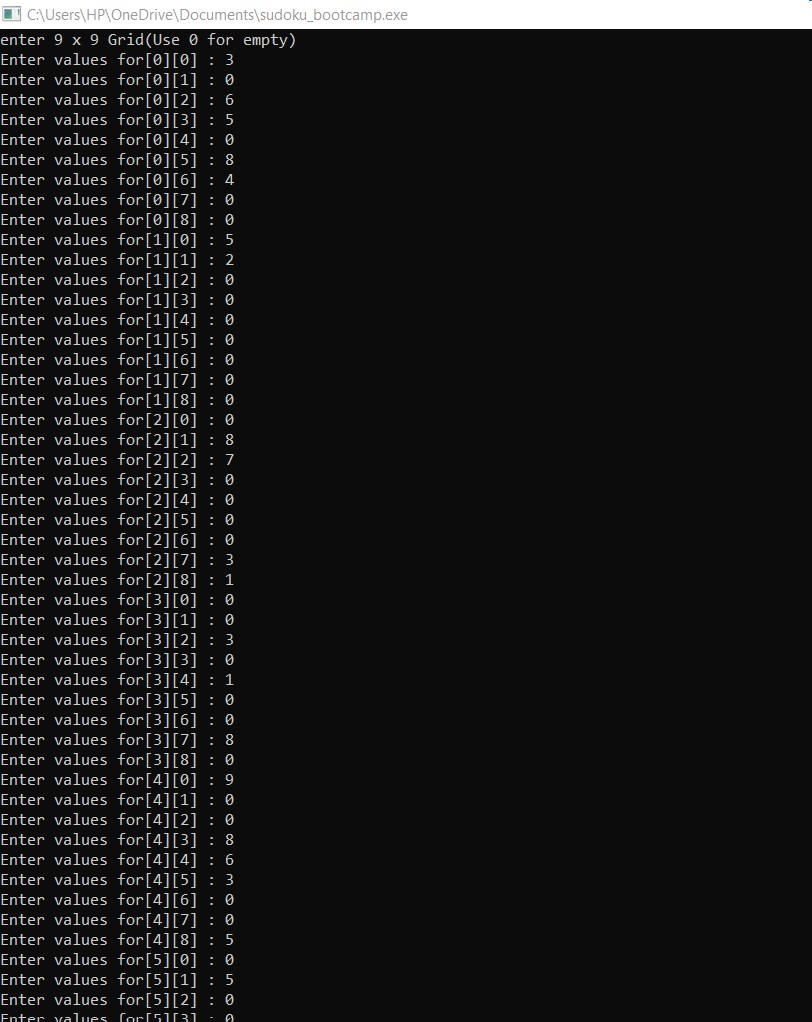
else

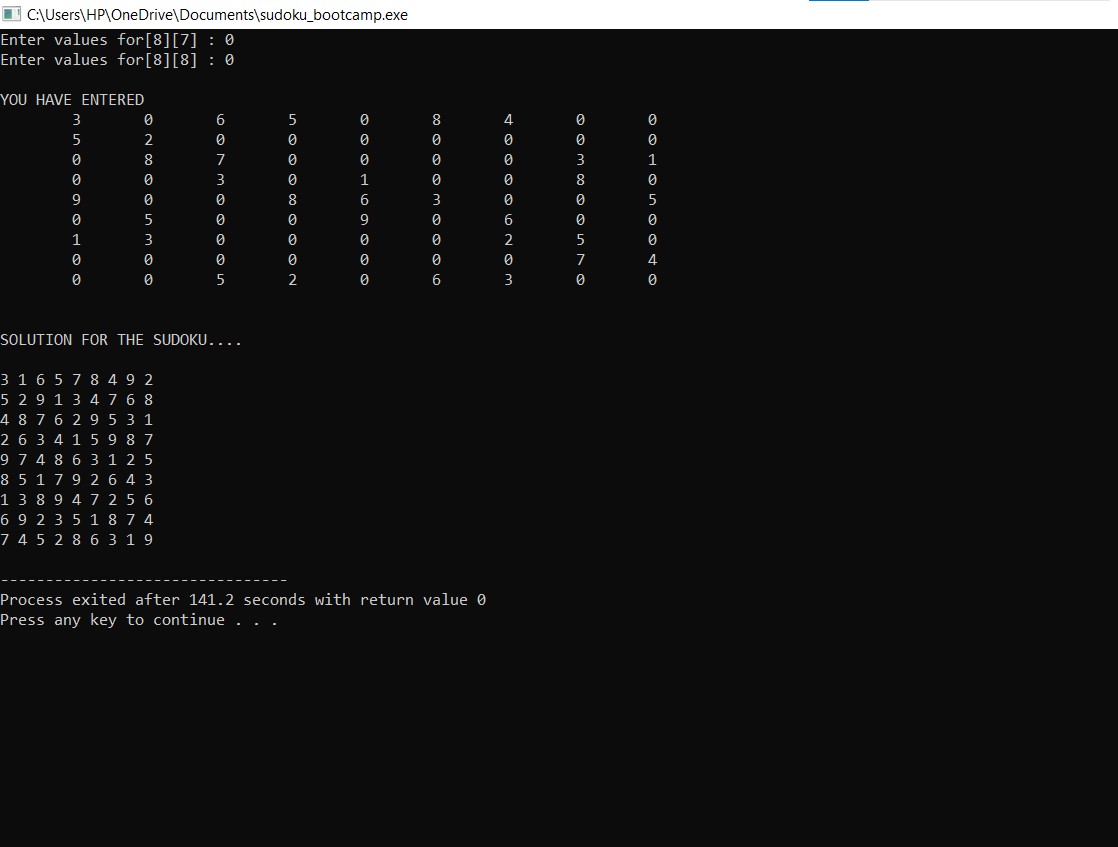
cout << "no solution exists " << endl;

return 0;

}

**Snapshot of Project**

****

****

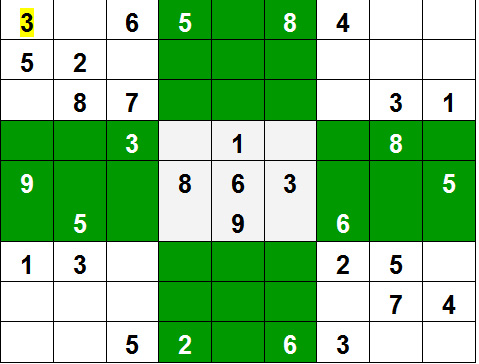
**Approach:** The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned position check if the matrix is safe or not. If safe print else recurs for other cases.

**Algorithm:**

1. Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true;
2. Create a recursive function that takes a grid and the current row and column index.
3. Check some base cases. If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false. The other base case is when the value of column is N, i.e j = N, then move to next row, i.e. i++ and j = 0.
4. if the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e. i, j+1. if the recursive call returns true then break the loop and return true.
5. if the current index is assigned then call the recursive function with index of next element, i.e. i, j+1

**Complexity Analysis:**

* **Time complexity:** O(9^(n\*n)).   
  For every unassigned index there are 9 possible options so the time complexity is O(9^(n\*n)).
* **Space Complexity:** O(n\*n).   
  To store the output array a matrix is needed.



**Literature Survey**

Smart Sudoku Solver

Summary

A Smart Sudoku Solver is presented that can solve unsolved Sudoku images with small amount of perspective. Also illumination changes across the images are taken care of. The algorithm can also give solution in cases of severe rotation such as when the Sudoku Puzzle is completely inverted. Since the scale of the image also varies from image to image, our algorithm efficiently manages these problems. Since there doesnt exist any standard Sudoku image dataset to test on, an own dataset of 100 test images was created and produced results on them. Since there are no standard images to compare to, it is difficult to quantify and compare results. However in general, for the 50 testing images were extracted from the web, found the algorithm to be 100translation a perspective skew of +- 45 degrees and robust to scale as long as the edge length of the Sudoku is at least 30commonly found in modern Sudoku images that give robust results for template matching.

**Advantages**

• The approach of applying Gaussian Blur and Adaptive Thresholding gave very good results in terms of grid detection. It removes any clutter outside the grid and isolates the Sudoku grid accurately and eases the rest of the algorithm.

• Back tracking algorithm takes less memory and time compared to Dancing links algorithm.

• A solution is guaranteed (as long as the puzzle is valid)

• Solving time is mostly unrelated to degree of difficulty

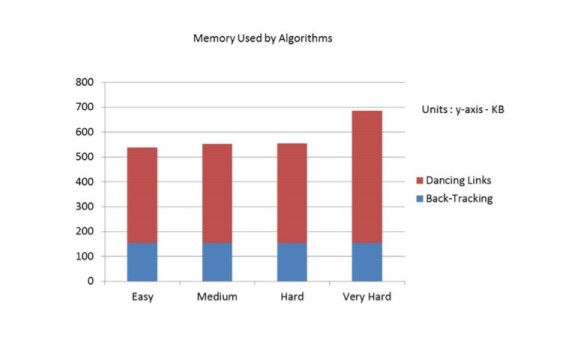
**Disadvantages**

• Hough transformation used to estimate the orientation of the lines prooved ineffective when the image is under heavy perspective skew, the lines are not necessarily straight, as a matter of fact, they are jagged thereby resulting in incorrect hough transform binning.

• It may be comparatively slow when compared to computer solution methods modeled after deductive methods.

**Comparison of different algorithms**

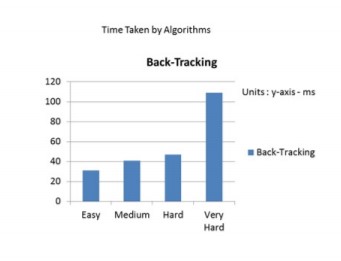
The following graph shows the comparison between backtracking and dancing links in terms of memory used for different difficulty levels of sudoku puzzle.



memory taken by different algorithms

**Time taken by the algorithms**

The following graphs shows the amount of time taken by the algorithm for different difficulty levels of sudoku puzzle.

****

**time taken by backtracking**

**References**

1. Pseudo-code for backtracking algorithm
   * <http://moritz.faui2k3.org/en/yasss>

2 www.geeksforgeeks.com