SUDOKU SOLVER

A PROJECT REPORT

Submitted by

SHAIK YASIR TAWFIQ [Reg No: RA2112704010016]

PONNURI ANIRUDDHA [Reg No: RA2112704010015]

Y SHABANYA KISHORE [Reg No:RA2112704010018]

Under the Guidance of

Dr. PAUL T SHEEBA

(Assistant Professor, Department of Data Science and Business Systems)

In partial fulfillment of the Requirements for the Degree

of

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND BUSINESS SYSTEMS

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

NOVEMBER 2022

Department of Data Science and Business Systems
SRM Institute of Science & Technology
Own Work* Declaration Form

To be completed by the student for all assessments

Degree/ Course : M TECH [INT] CSE WITH DATA SCIENCE

Student Name : SHAIK YASIR TAWFIQ

Registration Number : RA2112704010016

Title of Work : SUDOKU SOLVER

We hereby certify that this assessment complies with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g.fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

We understand that any false claim for this work will be penalized in accordance with theUniversity policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date forevery student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGYKATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this project report titled **"SUDOKU SOLVER"** is the bonafide work of **"SHAIK YASIR TAWFIQ [Reg No: RA2112704010016]"** and team who carried out the project work under my supervision. Certified further, that to the best ofmy knowledge the work reported herein does not form part of any other thesis or dissertationon the basis of which a degree or award was conferred on an earlier occasion for this or anyother candidate.

Dr. Paul T Sheeba
GUIDE
Assistant Professor
Dept. of DSBS

Dr. M. Lakshmi
HEAD OF THE DEPARTMENT
Dept. of DSBS

Signature of Internal Examiner

Signature of External Examiner

ABSTRACT

At this time, people all over the world are showing an increasing interest in playing puzzle games such as Sudoku. As a result of the game's growing popularity in a great number of countries, numerous game developers have attempted to create conundrums that are both more challenging and more engaging. Today, coverage of the game can be found in a significant number of books, newspapers, and online sources. In this article, we will discuss an algorithm that we have dubbed the "pencil-and-paper algorithm." This algorithm solves Sudoku puzzles by adhering to a few basic rules. The algorithm that is formulated using pencil-and-paper methods is based on human techniques. This indicates that the algorithm is designed with human perceptions in mind before being implemented. Consequently, we've decided to call this solver the pencil-and-paper algorithm. In order to determine whether or not the proposed algorithm is an improvement over the brute-force algorithm, this algorithm is first evaluated and then compared to the brute-force algorithm. The algorithm known as "brute force" is a generic one that can be utilised to solve any issue that may arise. This algorithm will come up with every conceivable solution before it finally settles on the correct one. The problem statement, the reason for doing this project, and the solution to the problem are discussed in the following subsections

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **DrT.V.Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work. We are incredibly grateful to our Head of the Department**, Dr M. Lakshmi** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work. We want to convey our thanks to our program coordinator **Dr.G.Vadivu,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her input during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor**, Dr.K.Shanthakumari,** Assistant Professor, DSBS, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr. Paul T Sheeba,** Assistant Professor, DSBS, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under his mentorship. He provided me with the freedom and support to explore the research topics of my interest. His passion for solving problems and making a difference in the world has always been inspiring. We sincerely thank the Data Science and Business Systems staff and students, SRM Instituteof Science and Technology, for their help during our project. Finally, we would like to thankparents, family members, and friends for their unconditional love, constant support, and encouragement.

Shaik Yasir Tawfiq

TABLE OF CONTENTS

ABBREVIATIONS

Sudoku	The word “Sudoku” is short for Su-ji wa dokushin ni kagiru (in Japanese), which means “the numbers must be single
Box	a region is a 9x9 box
Cell (Square)	is used to define the minimum unit of the Sudoku board.
Candidates	the number of possible values that can be placed into an empty square.
Clues	the given numbers in the grid at the beginning.
Grid	the Sudoku board consists of a form of matrix or windows

CHAPTER 1

INTRODUCTION

1.1 General

Kovacs details several of the brute force strategies that can be applied to the solution of Sudoku puzzles. The simplest approach, which is called "unconstrained grid," generates a solution to the puzzle through a process of randomization, and the computer then determines whether or not it is a valid solution. In the event that this does not work, the process is carried out once more until a solution is found. This algorithm can be easily applied, and it will find a valid solution for any problems that are presented to it because it will cycle through all of the possible solutions. However, according to Kovacs, the algorithm can be improved to make it more efficient. This method, on the other hand, may take a long time.

In most cases, the brute force algorithm will work its way through the empty squares, either filling them in with numbers taken from the available options or eliminating failed options when it reaches a "dead end." For instance, one way to solve a puzzle using brute force is to write the number "1" in the first empty square. If the digit can be placed there after checking the row, column, and box, the program will proceed to the next square and enter the digit "1" into that square. If the digit cannot be placed there, the program will exit. After the computer program determines that the number "1" cannot be used, the digit advances by one, meaning that it is now the number 2. When a square is found that does not allow any of the digits from 1 to 9, the program will retrace its steps and return to the previous square. The value contained within that square goes up by 1. The procedure is carried out once more until the correct digits have been entered into all 81 squares.

	a	b	c	d	e	f	g	h	i
1			9				7	2	8
2	2	7	8			3		1	
3		9					6	4	
4		5			6		2		
5			6				3		
6		1			5				
7	1			7		6		3	4
8				5		4			
9	7		9	1			8		5

Fig.1. *An example of a Sudoku puzzle.*

1.2 Problem Statement

The objective of the project is to conduct research into both the pencil-and-paper algorithm as well as the brute-force algorithm. The results of the comparison between these two approaches are shown later. It is anticipated that a productive approach for solving the Sudoku puzzles will be discovered here. In this article, we have attempted to implement the pencil-and-paper algorithm that simulates how a human being would solve the puzzle by employing some simple strategies that can be used to solve the majority of Sudoku puzzles. Using these strategies, we have been successful in solving a small percentage of the puzzles.

CHAPTER 2

LITERATURE REVIEW

Title 1. **Sudoku solver using mini grid-based backtracking**
(2018)

Sudoku' is a popular Japanese puzzle game that trains our logical mind. The word Sudoku means `the digits must remain single'. The Sudoku problem is important as it finds numerous applications in a variety of research domains with some sort of resemblance. Applications of solving a Sudoku instance are found in the fields of Steganography, Secret image sharing with necessary reversibility, Encrypting SMS, Digital watermarking, Image authentication, Image Encryption, and so and so forth. All the existing Sudoku solving techniques are primarily guess based heuristic or computation intensive soft computing methodology. They are all cell based, that is why very much time consuming.
Solving an instance of Sudoku problem is NP-complete. So it is unlikely to develop a deterministic polynomial time algorithm for solving a given Sudoku puzzle of size n×n, where n is any large number such that the square root of n is an integer. But incidentally when the value of n is bounded by some constant, solutions may be obtained in reasonable amount of time.
There are quite a few logic techniques that researchers use to solve this problem. Some are basic simple logic, some are more advanced. Depending on the difficulty of the puzzle, a blend of techniques may be needed in order to solve a puzzle. In fact, most computer-generated Sudoku puzzles rank the difficulty based upon the number of empty cells in the puzzle and how much effort is needed to solve each of them. Table 1 shows a comparison chart of the number of clues for different difficulty levels

Title 2. **The effect of guess choices on the efficiency of a backtracking algorithm in a Sudoku solver**
(2017)

There are several possible algorithms to automatically solve Sudoku boards; the most notable is the backtracking algorithm, that takes a brute-force approach to finding solutions for each board configuration. The performance of the backtracking algorithm is usually said to depend mainly on two implementation aspects: finding the next available empty cell in the board and finding the options of available legal number that are relevant to the given cell. While these pieces of the backtracking algorithm can vary in efficiency based on their implementation, tests show that the algorithm itself also relies on the statistical distribution of the guesses that it attempts to "plug in" to the board in every given cell. The backtracking algorithm uses an array of the legal numbers in the cell to attempt a solution before it moves on to the next cell. If a solution cannot be found, it backtracks and attempts to solve the board again with a different guess choice. The more errors the solver makes, the more backtracks it must perform, which decreases its overall efficiency and increases its effective runtime. Tests of the solving algorithm were performed using 195 base solutions with multiple initial board configurations were performed to analyze the difference in the algorithm performance by comparing the number of recursive backtracks between sequential and randomly distributed guesses. Analysis show that using values that

are given in a shuffled array significantly reduces the number of backtracks done by the solver and, as a result, improve the total effective efficiency of the algorithm as a whole

The most noticeable difference between the choices of guesses is the type of solutions that are produced. When the algorithm chooses its guesses in sequential order, the solutions given are deterministic; the same solution - with the same backtracks and recursions - will appear for the same initial board configurations. If the choice of guesses is shuffled, the solutions are varied, and so is the number of recursions and backtracks that the backtracking algorithm performs.

Title3 A Comparative Study on The Performance Characteristics of Sudoku Solving Algorithms (2021)

The experimental methodology takes an important role in deciding the fastest algorithm in terms of least time consumption for yielding solution to Sudoku. The algorithms (Brute Force, GA, SA, GRA, HS) discussed in the literature are implemented and test run in an IBM compatible PC with Dual Core Intel processor. The algorithms are fed with a set of 30 randomly chosen Sudoku puzzles from a collection of easy, medium and hard level puzzles. The time taken by each algorithm to solve each puzzle is observed and recorded. The design of such testing environment is set up as the algorithms under consideration have both heuristic and non-heuristic techniques where theoretical analysis only would not serve the purpose of exact performance analysis of these algorithms. The set used to test the algorithms are found to be a good mix of all types of puzzles thereby indicating a good set to judge the ability of algorithms.

CHAPTER 3

ALGORITHMS

3.1 Pencil-And-Paper Solver

There are several methods that are used by human players when playing Sudoku. However, it may be impossible to implement all these methods. It is found that the hidden single method or pair method are difficult to be applied in computer programming, since a human player has a better overview over the whole Sudoku board than the computer programming does. This is due to the fact that a human player is able to scan two rows or two columns in order to check whether a certain digit is allowed to be in an empty square in the box that is supposed to be filled up. Implementing the above task in computer programming causes significant time consumption.

The methods that are used in this algorithm are the following: --

- Unique missing
- Naked Single Method
- Backtracking

3.1.1 Unique missing

This method is useful when there is just only one empty square in a row, column or box. The digit that is missing can be placed in that empty square. A similar definition is that if eight of nine empty squares are filled in any row, column or box, then the digit that is missing can fill the only empty square. This method can be useful when most of the squares are filled, especially at the end of a solution. It can also be suitable when solving easy puzzle and this method is efficient to find solution in this case. In this algorithm, the method goes through all rows, columns and boxes separately. The method then checks if a single value has missed in any row, column or box and place the single digit in that specific square

3.1.2 Naked Single Method

The second method that is used in the pencil-and-paper algorithm is the Naked single method. This method checks every empty square in the Sudoku board and finds the square that can only take one single digit and the missing digit then is assigned to that square. Note that once the squares are filled by naked single digits other naked singles will appear. This process is

repeated until the method has found all empty squares with the needed corresponding one single value and complete the board

This method is a useful method when a human player solves the game. However, if the corresponding method is combined with the unique missing candidate method, then both the methods can solve the puzzles both in easy and medium levels quickly and more efficiently.

3.1.3 Backtracking

The unique missing method and the naked single method are able to solve all puzzles with easy and medium level of difficulties. In order to solve puzzles with even more difficult levels such as hard and evil the backtracking method has been used to complete the algorithm. A human player solves the puzzle by using simple techniques. If the puzzle is not solvable by using the techniques the player, then tries to fill the rest of the empty squares by guessing. The backtracking method, which is similar to the human strategy (guessing), is used as a help method to the pencil-and-paper algorithm. In other words, if the puzzle cannot be filled when using the unique missing method and the naked single method, the backtracking method will take the puzzle and fill the rest of empty squares. Generally, the backtracking method find empty square and assign the lowest valid number in the square once the content of other squares in the same row, column and box are considered. However, if none of the numbers from 1 to 9 are valid in a certain square, the algorithm backtracks to the previous square, which was filled recently. The above-mentioned methods are an appropriate combination to solve any Sudoku puzzles. The naked single method can find quickly single candidates to the empty squares that needed only one single value. Since the puzzle comes to its end solution the unique missing method can be used to fill rest of the puzzles. Finally, if either method fills the board the algorithm calls the backtracking method to fill the rest of the board

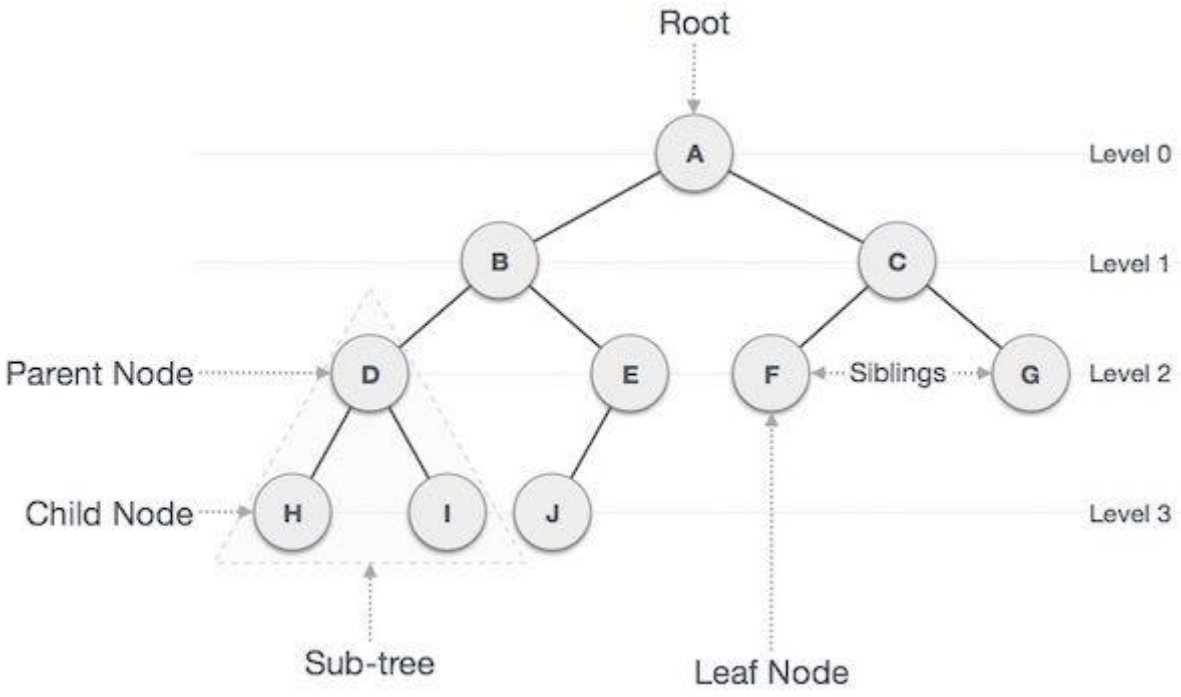
CHAPTER 4

SYSTEM DESIGN

Existing System

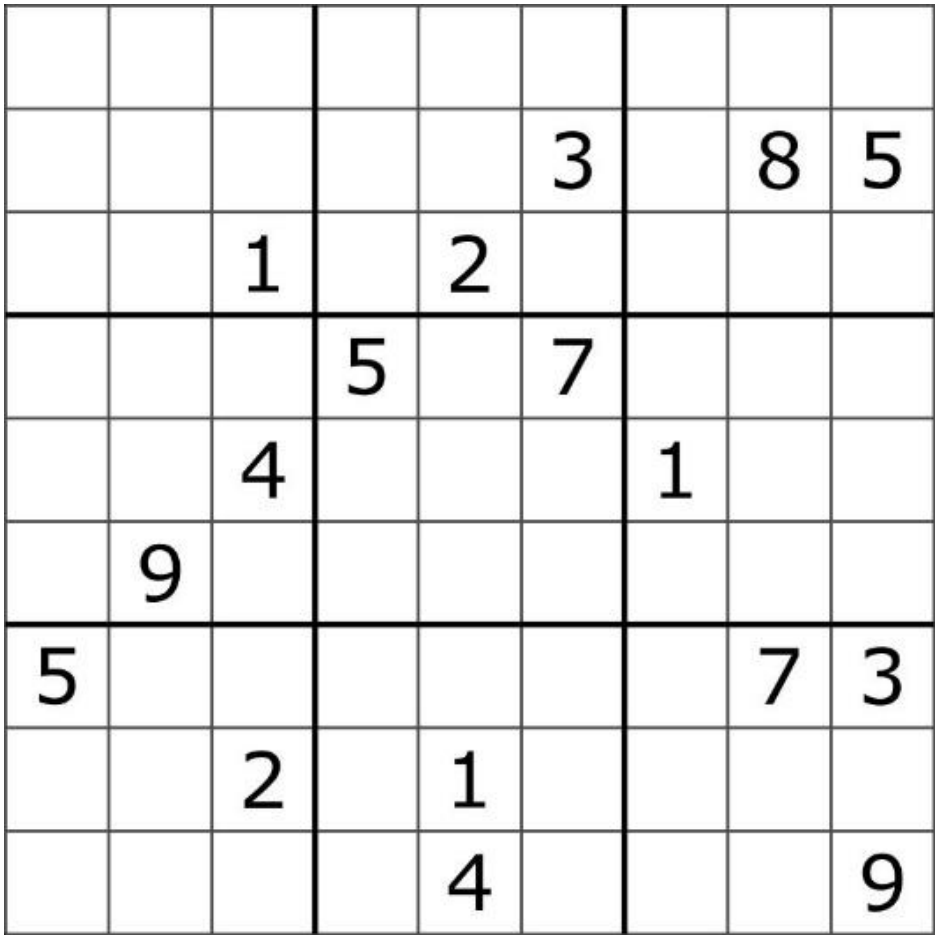
Puzzles based on the game Sudoku can be categorized as either an exact cover problem or, more accurately, an exact hitting set problem. Because of this, a sophisticated description of the problem and an effective solution are both possible. "The method of choice for rapid finding [measured in microseconds] of all possible solutions to Sudoku puzzles" is "the method of choice for modelling Sudoku as an exact cover problem and using an algorithm such as Knuth's Algorithm X and his Dancing Links technique." Using Gauss elimination in conjunction with column and row striking is an additional strategy that can be utilized as an alternative.

A Brute Force Algorithm is exactly what it sounds like: a straightforward method for solving a problem that relies on sheer processing power and attempting every alternative rather than complicated ways to improve efficiency. These algorithms are called "Brute Force" algorithms. The use of brute-force search is common when the size of the problem is relatively small or when there are problem-specific heuristics that can be applied to decrease the number of possible solutions to a level that is more manageable. The strategy is also employed in situations when the ease of implementation is prioritised over the amount of time it takes.



Let Q represent the 9x9 Sudoku matrix, N equal the numbers 1, 2, 3, 4, 5, 6, 7, 8, and 9, and X stand for any row, column, or block in the puzzle. In addition to providing symbols for filling out Q, N also provides an index set for each of the 9 components that make up any X. The elements q that are given in Q are representative of a partial function that goes from Q to N. The answer, denoted by R, is a total relation, and as such, it is a function. Because the rules of Sudoku mandate that the restriction of R to X must be a bijection, any partial solution C that is limited to an X is considered to be a partial permutation of N.

If we define T as "X: X is a row, column, or block of Q," then we can say that T contains 27 distinct elements. Either a partial permutation or a permutation on N can serve as the basis for an arrangement. Let Z represent the complete set of arrangements based on N. It is possible to reformulate a partial solution, denoted by the letter C, so that it includes the rules in the form of a composition of relations A (one-to-three) and B that demand compatible arrangements. The solution to the puzzle, as well as suggestions for new q to enter Q, come from prohibited arrangements, which are the complement of C in QxZ helpful tools in the calculus of relations are residual

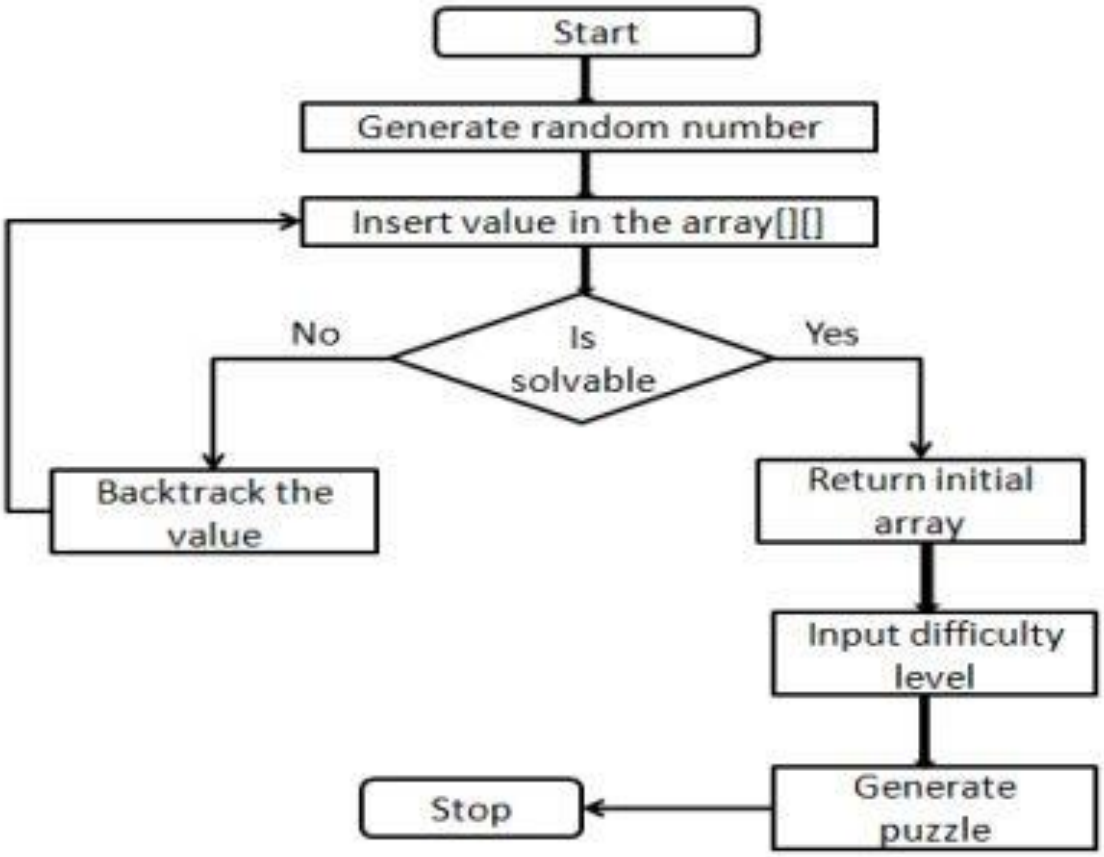


A sudoku decide work against brute force algorithm

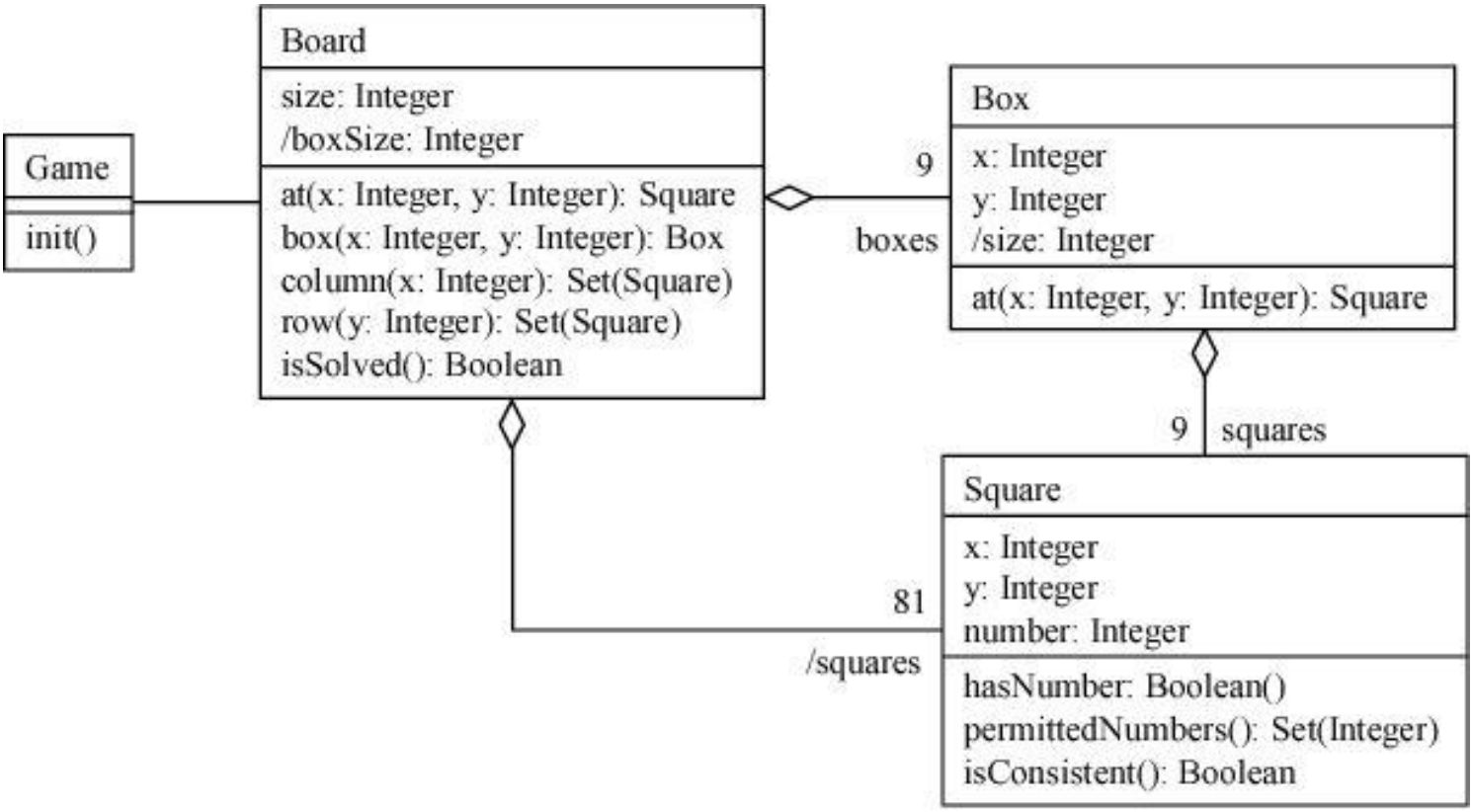
CHAPTER 5

Figures and Illustrative

- Data Flow Diagram



➢ Class Diagram



CHAPTER 6

SOURCE CODE

```
/*
 *=====================
 *   Sudoku Solver
 *=====================
 *
 * Objective - Takes in Sudoku puzzles and outputs the solution.
*/

#include<iostream>
#include<fstream>
using namespace std;


class SudokuFrame{

        int sudokuFrame[9][9];
        int editableFrame[9][9];


        public:SudokuFrame(){
                menu();
        }


        private:void menu(){
                cout<<"Welcome to Sudoku Solver!\n";
                cout<<"Before we start, you will have to input the puzzle into this program.\n\n";
```

```
        cout<<"You can either:-\n";
        cout<<"\t1. Input the puzzle by entering the values manually. \n";


        readFrameValues();

    }


    private:void readFrameValues(){
        cout<<"\nEnter the specified value when prompted.\n";
        cout<<"Enter 0 if cell is empty.\n\n";

        int rowIter, colIter;

        for(rowIter=0; rowIter<9; rowIter++){ //Iterating over cells to read vals.
            for(colIter=0; colIter<9; colIter++){
                int enteredValue;

                cout<<"Enter value for cell["<<rowIter+1<<"]["<<colIter+1<<"] --> ";
                cin>>enteredValue;

                if(!(enteredValue>=0 && enteredValue<=9)){ //Checking for bounds in input.
                    while(true){ //We loop until valid input is read from user.
                        cout<<"Oops! You seem to have entered a wrong value! Try again.\n";
                        cout<<"Enter value for cell ["<<rowIter+1<<"]["<<colIter+1<<"] --> ";
                        cin>>enteredValue;
```

```
                                        if(enteredValue>=0  &&  enteredValue<=9)
break;
                                    }
                                }

                                sudokuFrame[rowIter][colIter]=enteredValue;

                                if(enteredValue==0) editableFrame[rowIter][colIter]=0;
                                else editableFrame[rowIter][colIter]=1;
                            }
                            cout<<"-------\n";
                        }
                    }

                    public:void setCellValue(int row, int col, int num){
                        if(editableFrame[row][col]==0) sudokuFrame[row][col]=num;
                    }


                    public:int getCellValue(int row, int col){
                        int cellValue=sudokuFrame[row][col];
                        return cellValue;
                    }


                    public:int isEditable(int row, int col){
                        return (editableFrame[row][col]-1)*(-1);
                    }


                    public:void clearFrameFrom(int row, int col){
                        int jcount=0;
                        int rowIter, colIter;
```

```
            for(rowIter=row; rowIter<9; rowIter++){

                if(jcount==0) colIter=col;
                else colIter=0;

                for(; colIter<9; colIter++){
                    if(editableFrame[rowIter][colIter]==0)
sudokuFrame[rowIter][colIter]=0;
                }

                jcount++;

            }
        }


        public:void displayFrame(){


        cout<<"\033[0;36m++=================================
==++";
            int rowIter, colIter;

            for(rowIter=0; rowIter<9; rowIter++){
                int cellIter=1;

                cout<<"\n\033[0;36m||";
                for(colIter=0; colIter<9; colIter++){
                    if(cellIter==3){
                        cout<<"\033[0m
"<<sudokuFrame[rowIter][colIter]<<" ";
                        cout<<"\033[0;36m||";
                        cellIter=1;
                    }
```

```
                    else{
                        cout<<"\033[0m
"<<sudokuFrame[rowIter][colIter]<<"  ";
                        cellIter++;
                    }
                }

                if(rowIter%3!=2)    cout<<"\n\033[0;36m++-----------++-----------
++-----------++";
                else
cout<<"\n\033[0;36m++=====================================++
+";
            }

        }

};


class Possibilities{

    struct node{
        int value;
        struct node* next;
    };

    typedef struct node* Node;

    Node head;
    Node pos;


    public:Possibilities(){
```

```
        head=new struct node;
        head->next=NULL;
}

public:~Possibilities(){
        destroy();
}


public:void append(int number){
        Node temp=new struct node;

        temp->value=number;
        temp->next=NULL;

        pos=head;
        while(pos!=NULL){
                if(pos->next==NULL){
                        pos->next=temp;
                        break;
                }
                pos=pos->next;
        }
}

public:int operator[](int index){
        int count=0;
        pos=head->next;

        while(pos!=NULL){
                if(count==index)
                        return pos->value;
                pos=pos->next;
                count++;
```

```
        }

        return -1;
}


public:void print(){
        pos=head->next;
        while(pos!=NULL){
                cout<<pos->value<<",";
                pos=pos->next;
        }
        cout<<"\b";
}

public:int length(){
        pos=head->next;
        int count=0;

        while(pos!=NULL){
                count++;
                pos=pos->next;
        }

        return count;
}


public:void copy(Possibilities possibilities){
        int oldLength=possibilities.length();
        int iter=0;

        pos=head;
        for(iter=0; iter<oldLength; iter++){
```

```
                    Node temp=new struct node;

                    temp->next=NULL;
                    temp->value=possibilities[iter];

                    pos->next=temp;
                    pos=pos->next;
                }
        }


        private:void destroy(){
                Node temp;
                pos=head;
                while(pos!=NULL){
                        temp=pos;
                        pos=pos->next;
                        delete temp;
                }
        }

};



class SudokuSolver{

        int recursiveCount; //Stats variable
        SudokuFrame frame; //The frame object


        public:SudokuSolver(){
                recursiveCount=0;
```

```
        cout<<"\nCalculating possibilities...\n";
        cout<<"Backtracking across puzzle....\n";
        cout<<"Validating cells and values...\n\n";

        solve();
        cout<<"QED. Your puzzle has been solved!\n\n";
        displayFrame();

        cout<<"\n\n";
}

private:bool cellValueValid(int row, int col, int currentValue){
        int rowIter, colIter;

        //Checking if value exists in same column
        for(rowIter=0; rowIter<9; rowIter++){
                if(rowIter!=row){
                        int comparingValue=frame.getCellValue(rowIter,col);
                        if(comparingValue==currentValue) return false;
                }
        }

        //Checking if value exists in same row
        for(colIter=0; colIter<9; colIter++){
                if(colIter!=col){
                        int comparingValue=frame.getCellValue(row,colIter);
                        if(comparingValue==currentValue) return false;
                }
        }

        //Checking if value exists in the same 3x3 square block
        if(ThreeByThreeGridValid(row,col,currentValue)==false) return false;

        return true;
```

```
    }


    private:bool ThreeByThreeGridValid(int row, int col, int currentValue){
        int rowStart=(row/3)*3;
        int rowEnd=(rowStart+2);

        int colStart=(col/3)*3;
        int colEnd=(colStart+2);

        int rowIter, colIter;

        for(rowIter=rowStart; rowIter<=rowEnd; rowIter++){
            for(colIter=colStart; colIter<=colEnd; colIter++){
                if(frame.getCellValue(rowIter,colIter)==currentValue)
return false;
            }
        }

        return true;
    }


    private:Possibilities getCellPossibilities(int row, int col){
        int iter=0;

        Possibilities possibilities;

        for(iter=1; iter<=9; iter++){
            if(cellValueValid(row,col,iter)==true)
                possibilities.append(iter);
        }

        return possibilities;
```

```
}

private:int singleCellSolve(int row, int col){

    statsIncrement(); //This is used to see how many times the func is called.

    if(frame.isEditable(row,col)==true){

        Possibilities possibilities;
        possibilities.copy(getCellPossibilities(row,col));

        int posLength=possibilities.length();
        int posIter=0, newRow=row, newCol=col;

        for(posIter=0; posIter<posLength; posIter++){ //We iter over the possible values
            int possibility=possibilities[posIter];
            frame.setCellValue(row,col,possibility);

            //We now increment the col/row values for the next recursion
            if(col<8) newCol=col+1;
            else if(col==8){
                if(row==8) return 1; //this means success
                newRow=row+1;
                newCol=0;
            }

            {

                if(singleCellSolve(newRow,newCol)==0){ //If wrong, clear frame and start over
                    frame.clearFrameFrom(newRow,newCol);
                }
```

```
                    else return 1;

                }

            }

            return 0;

        }
        else{

            int newRow=row, newCol=col;

            //Same incrementing of the col/row values
            if(col<8) newCol=col+1;
            else if(col==8){
                if(row==8) return 1;
                newRow=row+1;
                newCol=0;
            }

            return singleCellSolve(newRow,newCol);

        } //The else block ends here

}

private:void solve(){
        int success=singleCellSolve(0,0);
}

private:void displayFrame(){
        frame.displayFrame();
}
```

```
        private:void statsIncrement(){
                recursiveCount++;
        }


        public:void statsPrint(){
                cout<<"\nThe singleCellSolve() function was recursively called "<<recursiveCount<<" times.\n";
        }

};

int main(){
        SudokuSolver ss;
        return 0;
}
```

CHAPTER 7

OUTPUT

```
Microsoft Windows [Version 10.0.22621.898]
(c) Microsoft Corporation. All rights reserved.

D:\pFiles\Sudoku-Solver>cd "d:\pFiles\Sudoku-Solver"

d:\pFiles\Sudoku-Solver>cd "d:\pFiles\Sudoku-Solver\" && g++ sudoku-solver_1.cpp -o sudoku-solver_1 && "d:\pFiles\Sudok
u-Solver\"sudoku-solver_1
Welcome to Sudoku Solver!
Before we start, you will have to input the puzzle into this program.

You can either:-
        1. Input the puzzle by entering the values manually.

Enter the specified value when prompted.
Enter 0 if cell is empty.

Enter value for cell[1][1] --> 0
Enter value for cell[1][2] --> 0
Enter value for cell[1][3] --> 3
Enter value for cell[1][4] --> 4
Enter value for cell[1][5] --> 8
Enter value for cell[1][6] --> 0
Enter value for cell[1][7] --> 9
Enter value for cell[1][8] --> 7
Enter value for cell[1][9] --> 0
-------
```

```
-------
Enter value for cell[6][1] --> 0
Enter value for cell[6][2] --> 3
Enter value for cell[6][3] --> 0
Enter value for cell[6][4] --> 0
Enter value for cell[6][5] --> 0
Enter value for cell[6][6] --> 0
Enter value for cell[6][7] --> 0
Enter value for cell[6][8] --> 0
Enter value for cell[6][9] --> 0
-------
Enter value for cell[7][1] --> 60
Oops! You seem to have entered a wrong value! Try again.
Enter value for cell [7][1] --> 6
Enter value for cell[7][2] --> 0
Enter value for cell[7][3] --> 0
Enter value for cell[7][4] --> 0
Enter value for cell[7][5] --> 9
Enter value for cell[7][6] --> 0
Enter value for cell[7][7] --> 3
Enter value for cell[7][8] --> 0
Enter value for cell[7][9] --> 0
-------
```

```
Enter value for cell[2][1] --> 0
Enter value for cell[2][2] --> 0
Enter value for cell[2][3] --> 0
Enter value for cell[2][4] --> 9
Enter value for cell[2][5] --> 0
Enter value for cell[2][6] --> 0
Enter value for cell[2][7] --> 6
Enter value for cell[2][8] --> 0
Enter value for cell[2][9] --> 2
-------
Enter value for cell[3][1] --> 0
Enter value for cell[3][2] --> 0
Enter value for cell[3][3] --> 9
Enter value for cell[3][4] --> 0
Enter value for cell[3][5] --> 3
Enter value for cell[3][6] --> 0
Enter value for cell[3][7] --> 0
Enter value for cell[3][8] --> 0
Enter value for cell[3][9] --> 8
-------
Enter value for cell[4][1] --> 0
Enter value for cell[4][2] --> 0
Enter value for cell[4][3] --> 0
Enter value for cell[4][4] --> 0
Enter value for cell[4][5] --> 0
Enter value for cell[4][6] --> 0
Enter value for cell[4][7] --> 0
Enter value for cell[4][8] --> 6
```

```
Calculating possibilities...
Backtracking across puzzle....
Validating cells and values...

QED. Your puzzle has been solved!

++=======================================++
|| 1   6   3 || 4   8   2 || 9   7   5 ||
++-----------++-----------++-----------++
|| 7   4   8 || 9   1   5 || 6   3   2 ||
++-----------++-----------++-----------++
|| 2   5   9 || 6   3   7 || 4   1   8 ||
++=======================================++
|| 5   8   2 || 3   4   1 || 7   6   9 ||
++-----------++-----------++-----------++
|| 4   7   6 || 2   5   9 || 1   8   3 ||
++-----------++-----------++-----------++
|| 9   3   1 || 7   6   8 || 5   2   4 ||
++=======================================++
|| 6   2   7 || 8   9   4 || 3   5   1 ||
++-----------++-----------++-----------++
|| 8   1   4 || 5   7   3 || 2   9   6 ||
++-----------++-----------++-----------++
|| 3   9   5 || 1   2   6 || 8   4   7 ||
++=======================================++
```

CHAPTER 8

CONCLUSION

The results of this research indicate that the pencil-and-paper algorithm is a viable strategy that may be used to solve any Sudoku puzzle. In comparison to the brute force approach, the algorithm is a more appropriate method that can find a solution in a shorter amount of time and with greater efficiency. The suggested algorithm is capable of quickly solving such puzzles, regardless of their degree of difficulty, in a relatively short amount of time (less than one second).

The results of the tests have shown that the performance of the pencil-and-paper algorithm is superior to the performance of the brute force technique in terms of the amount of processing time required to solve any puzzle.

The brute force strategy appears to be a helpful method for solving any and all Sudoku puzzles, and it can ensure the discovery of at least one solution. On the other hand, this technique is inefficient due to the fact that the difficulty level has no bearing on how the algorithm works. To put it another way, the algorithm does not make use of any clever tactics when attempting to solve the riddles. It is a time-consuming process that ultimately results in an inefficient puzzle solver because this algorithm investigates all of the potential answers to the riddle until it finds one that is correct. The most significant benefit of utilising the method is, as has been mentioned previously, the capacity to solve any puzzles, with a solution being absolutely assured to be found.

CHAPTER 9

FUTURE ENHANCEMENTS

In order to improve the performance of the pencil-and-paper method, additional investigation into the topic is required. Other human tactics represent a potential solution that could be used (x-wings, swordfish, etc.).

Other possibilities include determining whether it is possible to implement an algorithm that is based solely on human strategies, so that the pencil-and-paper algorithm does not involve any other algorithms, and ensuring that these strategies can solve any puzzle, regardless of its degree of difficulty.

CHAPTER 11

REFERENCES

i. A SAT-based Sudoku solver
T Weber - LPAR, 2005 - cs.miami.edu

ii. The tu delft sudoku solver on fpga
K Van Der Bok, M Taouil, P Afratis… - … Conference on Field …, 2009 – Ieeexplore.ieee.org

iii. Sudoku solver by q'tron neural networks
TW Yue, ZC Lee - International Conference on Intelligent Computing, 2006 - Springer

iv. All-optical Sudoku solver with photonic spiking neural network
S Gao, S Xiang, Z Song, Y Han, Y Hao - Optics Communications, 2021 – Elsevier

v. Recognition of numbers and position using image processing techniques for solving sudoku puzzles
PJ Simha, KV Suraj, T Ahobala - IEEE-international conference …, 2012 - ieeexplore.ieee.org

vi. Spiking analog VLSI neuron assemblies as constraint satisfaction problem solvers
J Binas, G Indiveri, M Pfeiffer - 2016 IEEE International …, 2016 - ieeexplore.ieee.org

APPENDIX:

https://github.com/RA2112704010016/DSA_MINI_PROJECT

https://github.com/RA2112704010016