DISCRETE MATHEMATICS

SUDOKU SOLVER AND VALIDATOR

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

SUBMITTED BY:
Ayush kumar (2K19/SE/026)
Ayush Sharma (2K19/SE/027)

SUBMITTED TO:
SURYA GIRI SIR

ACKNOWLEDGEMENT

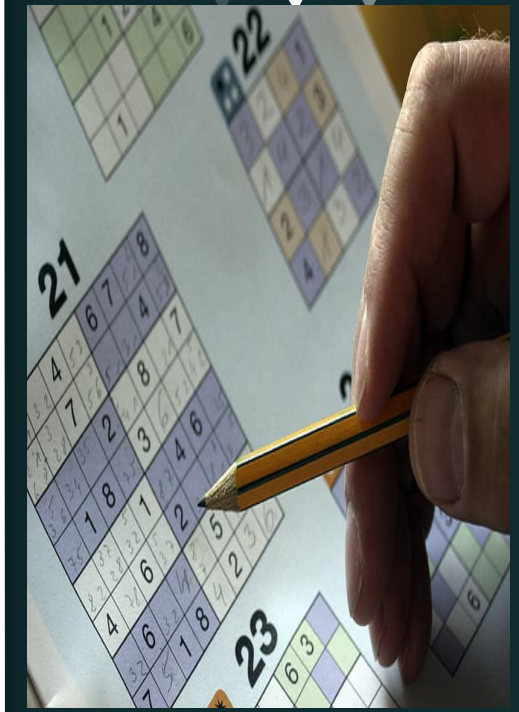
At the very outset of this report, we would like to extend our sincere and heartfelt obligation towards all the personages who have guided us with the project.

A special thanks to SURYA GIRI SIR for teaching us the subject “DISCRETE MATHEMATICS”. He helped us visualize the subject and to find its applications in real life. He supervised us with the intricacies of this project. He also offered many relevant and productive recommendations for the project, for which we are very grateful.

Finally, a thank you to all our family and friends who helped us with the project during such difficult times and gave worthy ideas.

MOTIVATION

We all love solving puzzles, crosswords but often get tired when we are unable to get the solution. So, in our project, we have designed a Sudoku solver and validator mainly using Bool functions in C++. The use of Bool functions increases the execution speed of the Sudoku solver significantly and also makes it easier to understand.



INTRODUCTION

Sudoku is a very popular puzzle that consists of placing several numbers in a squared grid according to some simple rules. This involves logic and combinatorial number placements by applying satisfiability (SAT/SMT).We all have played Sudoku at some time in our lives, especially during the current lockdown, and have scratched our heads trying to find the answers. Therefore, this project not only hopes to solve the Sudoku but in an easier, faster, and interesting way.

AIM

Designing a sudoku solver and validator using boolean algebra. In this project, we aim to apply Boolean algebra to solve a sudoku puzzle, given by the user. The sudoku validator will tell the user if a given fully solved sudoku is a valid solution or not.

—

Approach:

The approach of this project is to make a Sudoku solver that is much better in the terms of time complexity, space complexity and interesting using Boolean algebra, after a thorough research on this topic. It should outperform the existing popular approaches, making our Sudoku solver superior than the others.

Technology Stack:

In this project, most of our research will be done on paper, understanding and applying different approaches, however to test all that and come up to the best possible solution, we will be using python, PHP, AI.

—

Deliverables:

• An accurate implementation of the approach best suited after going through a research and brainstorming.

• Image soft tested cases and output.
• Final detailed documentedTimeline:

PRELIMINARY KNOWLEDGE:

- ➔ Boolean Algebra
- ➔ Sudoku Rules
- ➔ Basic C++ experience

ALGORITHM USED:

FOR SOLVER:

1. Start
2. We will start with an empty cell.
3. We will generate a list of all possible values that can be filled in that cell.
4. We traverse over this list and start with the first value and place it in the required cell.
5. We move on to the next cell. We again generate a list of possibilities. However, if no list can be generated, then this means that there is something wrong with the value of the previous cell. We then move back to the previous cell and place the next value on the generated list in the cell now. We repeat this step until the current cell has a valid value placed inside it.
6. We stop when we reach the last cell. And have placed a valid value.
7. Stop

IMPLEMENTATION DETAILS

Input the values for each cell for the sudoku you wish to be solved. Input 0 for empty cells.

The function “setCellValue” then passes these values to their respective cells. Not just it takes the value but also checks if the puzzle is valid by checking if the value occurs in that row, column, or 3X3 grid or not. This is done using the function cellValueValid and ThreeByThreeGirdValid

Then the function SingleCellSolve is called. This is the main function that solves the puzzle by each cell one by one.We will only go to the cells which have 0 in them and are, thus, editable.

After this, we have made a class called Possibilities which will calculate all the possible values for a cell. This class stores the values in a linked list.

From the possible values, we check for each value if the value is valid or not. By valid, we mean if that possible value occurs in that row, column, or 3X3 grid or not. This is done using the function cellValueValid and ThreeByThreeGirdValid.

We assign each cell a value one by one after checking its validity.

However, since each cell may have more than 1 value, a cell once assigned a value might change later. Once fixed, that value is later on eliminated. This process is done by backtracking by checking the validity of previous cells again and again after moving on to the other.

★For Sudoku Solver:

We developed a sudoku solver for the most popular 9*9 grid with nine rows and nine columns and the 9*9 grid is divided into nine 3*3 sub-grids.
Boolean algebra is much faster than arithmetic algebra. We have used boolean algebra to determine the possible value of a cell. We have used a 2-D array to store the value of each cell. We have made almost all the utility functions, boolean function using the bool keyword.
Initially, the program would solve the puzzle by placing the digit “1” in the first empty cell and checking if it is allowed to be there. If there are no violations (checking row, column, and box constraints) then the algorithm advances to the next cell and places a “1” in that cell. When checking for violations, if it is discovered that the “1” is not allowed, the value is changed to “2”. If a cell is discovered where none of the 9 digits is allowed, then the algorithm leaves that cell blank and moves back to the previous cell. The value in that cell is then incremented by one. This is repeated until an allowed value in the last empty cell is discovered.

—

➢ Vectors

Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted. This usually helps as sometimes we are unfamiliar with the length of values we have to store in them. As well as, they help in saving space.

➢ Linked Lists

A linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list. This helps us in traversing back and forth easily. We have made a class called Possibilities which will calculate all the possible values for a cell. This class stores the values in a linked list.

➢ Classes

A class is a user-defined data type that we can use in our program, and it works as an object constructor or a "blueprint" for creating objects . We created a class called Possibilities, SudokuSolver, and SudokuValidator and their functionality will be explained later.

—

➢ Functions

A function is a group of statements that together perform a task. In our code, we have made multiple functions that perform various tasks. For example, functions like readFrameValues(), setCellValue(), and isEditable() are used in our code that used to perform tasks like reading inputs, setting the values to their respective cells, to check if a cell can be edited or not, etc.

➢ Backtracking

Backtracking is an algorithm for finding all (or some) of the solutions to a problem that incrementally builds candidates to the solution(s). As soon as it determines that a candidate cannot possibly lead to a valid solution, it abandons the candidate. Our algorithm visits the empty cells in sequential order, filling in digits sequentially, or backtracking when no digit can be filled without violating any rule.

ALGORITHM USED:

FOR VALIDATOR:

1. Start
2. The values in all the cells are traversed one by one and checked to see whether they are in the range of 1-9. If not, then the puzzle is invalid.
3. Every row is checked to see if it contains 1-9 at least and at most once. If not, then the puzzle is invalid.
4. Every column is checked to see if it contains 1-9 at least and at most once. If not, the solution is invalid.
5. Every 3x3 grid is checked to see if it contains 1-9 at least and at most once. If not, the solution is invalid.
6. If all the criteria have been satisfied, the solution is valid.
7. Stop

—

For Sudoku Validator:

In this section, we validate solutions for 9x9 Sudoku puzzles using boolean functions. This program takes input from the user and then validates the puzzle and then displays whether it is a valid solution or not. This validator uses that same algorithm that we have used in the solver, to check the validity of the values inserted in the cells.

CODE

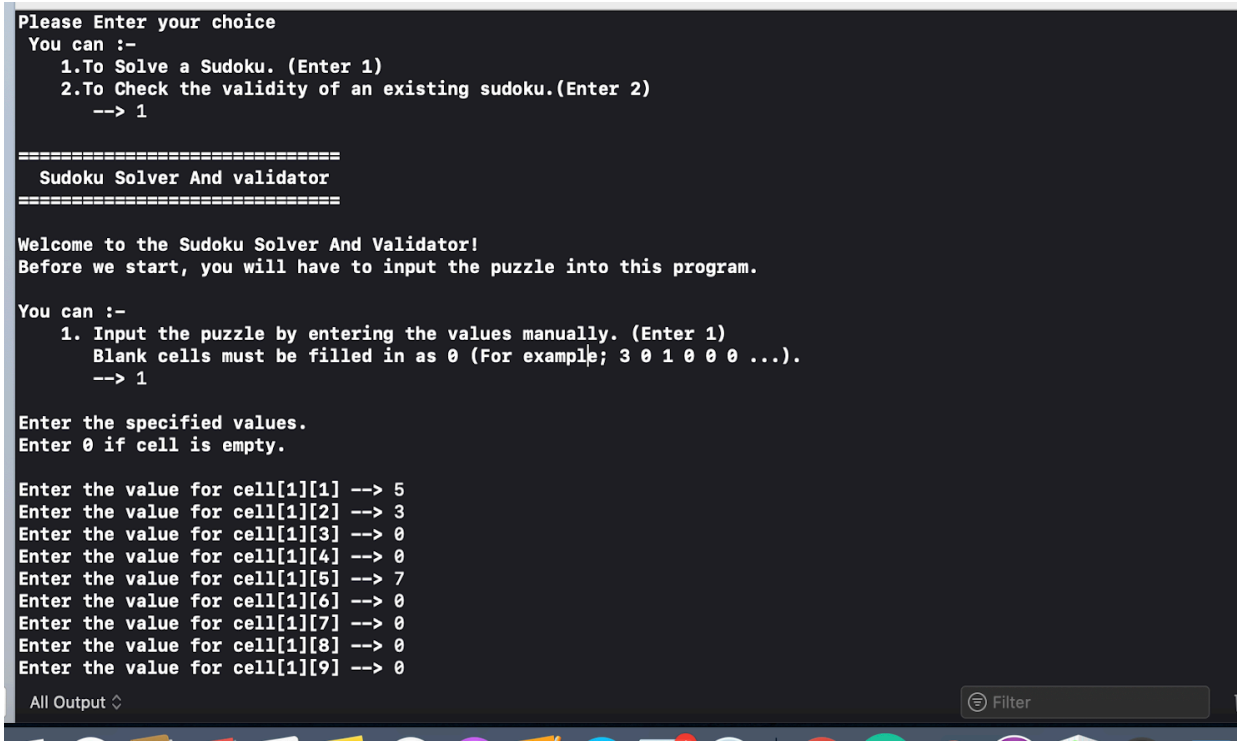
```
#include<iostream>
#include<fstream>
using namespace std;

class SudokuFrame{

    int sudokuFrame[9][9];
    int editableFrame[9][9];
    public:SudokuFrame(){
        display_menu();
    }


    private:void display_menu(){

        cout<<"\n============================\n";
        cout<<"  Sudoku Solver And validator\n";
        cout<<"============================\n\n";

        cout<<"Welcome to the Sudoku Solver And Validator!\n";
        cout<<"Before we start, you will have to input the puzzle into this program.\n\n";
        cout<<"You can :-\n";
        cout<<"\t1. Input the puzzle by entering the values manually. (Enter 1)\n";

        cout<<"\t   Blank cells must be filled in as 0 (For example; 3 0 1 0 0 0 ...).\n";
        cout<<"\t   --> ";

        int option;
        cin>>option;

        if(option==1) readFrameValues();
        else{
            while(true){
                cout<<"\nOops!! You have entered an invalid option. Try again.\n";
                cout<<"\t   --> ";
                cin>>option;

                if(option==1) readFrameValues();
                else continue;
```

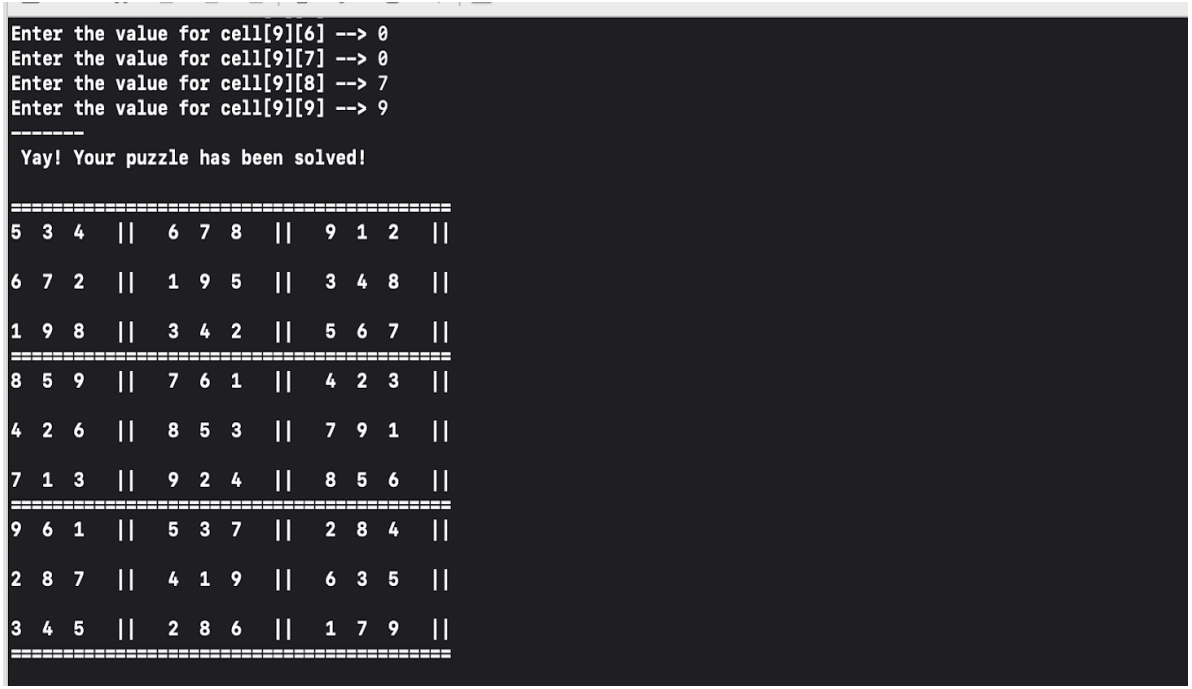
```
    private:void readFrameValues(){
        cout<<"\nEnter the specified values.\n";
        cout<<"Enter 0 if cell is empty.\n\n";

        int row_val, col_val;

        for(row_val=0; row_val<9; row_val++){
            for(col_val=0; col_val<9; col_val++){
                int value;

                cout<<"Enter the value for cell["<<row_val+1<<"]["<<col_val+1<<"] --> ";
                cin>>value;

                if(!(value>=0 && value<=9)){
                    while(true){
                        cout<<"Oops! You have entered a wrong value! Try again.\n";
                        cout<<"Enter the value for cell ["<<row_val+1<<"]["<<col_val+1<<"] --> ";
                        cin>>value;

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

                sudokuFrame[row_val][col_val]=value;

                if(value==0) editableFrame[row_val][col_val]=0;
                else editableFrame[row_val][col_val]=1;
            }
            cout<<"-------\n";
        }
    }


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


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

```
100         return (editableFrame[row][col]-1)*(-1);
101     }
102 
103 
104     public:void clearFrameFrom(int row, int col){
105         int jcount=0;
106         int row_val, col_val;
107 
108         for(row_val=row; row_val<9; row_val++){
109 
110             if(jcount==0) col_val=col;
111             else col_val=0;
112 
113             for(; col_val<9; col_val++){
114                 if(editableFrame[row_val][col_val]==0) sudokuFrame[row_val][col_val]=0;
115             }
116 
117             jcount++;
118 
119         }
120     }
121 
122     public:void displayFrame(){
123 
124         cout<<"========================================";
125         int row_val, col_val;
126 
127         for(row_val=0; row_val<9; row_val++){
128             int cellIter=1;
129 
130             cout<<"\n";
131             for(col_val=0; col_val<9; col_val++){
132                 if(cellIter==3){
133                     cout<<sudokuFrame[row_val][col_val]<<"   ||   ";
134                     cellIter=1;
135                 }
136                 else{
137                     cout<<sudokuFrame[row_val][col_val]<<"  ";
138                     cellIter++;
139                 }
140             }
141 
142             if(row_val%3!=2) cout<<"\n";
```

```
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;
```

RESULTS


Please Enter your choice
You can :-
1.To Solve a Sudoku. (Enter 1)
2.To Check the validity of an existing sudoku.(Enter 2)
--> 1
==========
Sudoku Solver And validator
==========
Welcome to the Sudoku Solver And Validator!
Before we start, you will have to input the puzzle into this program.
You can :-
1. Input the puzzle by entering the values manually. (Enter 1)
Blank cells must be filled in as 0 (For example; 3 0 1 0 0 0 ...).
--> 1
Enter the specified values.
Enter 0 if cell is empty.
Enter the value for cell[1][1] --> 5
Enter the value for cell[1][2] --> 3
Enter the value for cell[1][3] --> 0
Enter the value for cell[1][4] --> 0
Enter the value for cell[1][5] --> 7
Enter the value for cell[1][6] --> 0
Enter the value for cell[1][7] --> 0
Enter the value for cell[1][8] --> 0
Enter the value for cell[1][9] --> 0
All Output
Filter


Enter the value for cell[9][6] --> 0
Enter the value for cell[9][7] --> 0
Enter the value for cell[9][8] --> 7
Enter the value for cell[9][9] --> 9

Yay! Your puzzle has been solved!
5 3 4 || 6 7 8 || 9 1 2 ||
6 7 2 || 1 9 5 || 3 4 8 ||
1 9 8 || 3 4 2 || 5 6 7 ||
8 5 9 || 7 6 1 || 4 2 3 ||
4 2 6 || 8 5 3 || 7 9 1 ||
7 1 3 || 9 2 4 || 8 5 6 ||
9 6 1 || 5 3 7 || 2 8 4 ||
2 8 7 || 4 1 9 || 6 3 5 ||
3 4 5 || 2 8 6 || 1 7 9 ||

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

Your puzzle is valid!
```

Sudoku Validator:

```
Please Enter your choice
 You can :-
    1.To Solve a Sudoku. (Enter 1)
    2.To Check the validity of an existing sudoku.(Enter 2)
       --> 2

============================
  Sudoku Solver And validator
============================

Welcome to the Sudoku Solver And Validator!
Before we start, you will have to input the puzzle into this program.

You can :-
    1. Input the puzzle by entering the values manually. (Enter 1)
       Blank cells must be filled in as 0 (For example; 3 0 1 0 0 0 ...).
       --> 1

Enter the specified values.
Enter 0 if cell is empty.

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

```
-------
Enter the value for cell[2][1] --> 9
Enter the value for cell[2][2] --> 6
Enter the value for cell[2][3] --> 5
Enter the value for cell[2][4] --> 3
Enter the value for cell[2][5] --> 2
Enter the value for cell[2][6] --> 7
Enter the value for cell[2][7] --> 1
Enter the value for cell[2][8] --> 4
Enter the value for cell[2][9] --> 8
-------
Enter the value for cell[3][1] --> 3
Enter the value for cell[3][2] --> 4
Enter the value for cell[3][3] --> 1
Enter the value for cell[3][4] --> 6
Enter the value for cell[3][5] --> 8
Enter the value for cell[3][6] --> 9
Enter the value for cell[3][7] --> 7
Enter the value for cell[3][8] --> 5
Enter the value for cell[3][9] --> 2
-------
Enter the value for cell[4][1] --> 5
Enter the value for cell[4][2] --> 9
Enter the value for cell[4][3] --> 3
Enter the value for cell[4][4] --> 4
Enter the value for cell[4][5] --> 6
Enter the value for cell[4][6] --> 8
Enter the value for cell[4][7] --> 2
Enter the value for cell[4][8] --> 7
Enter the value for cell[4][9] --> 1
-------
All Output ◇                                   Filter
```

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

All Output

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

Your puzzle is valid!
```

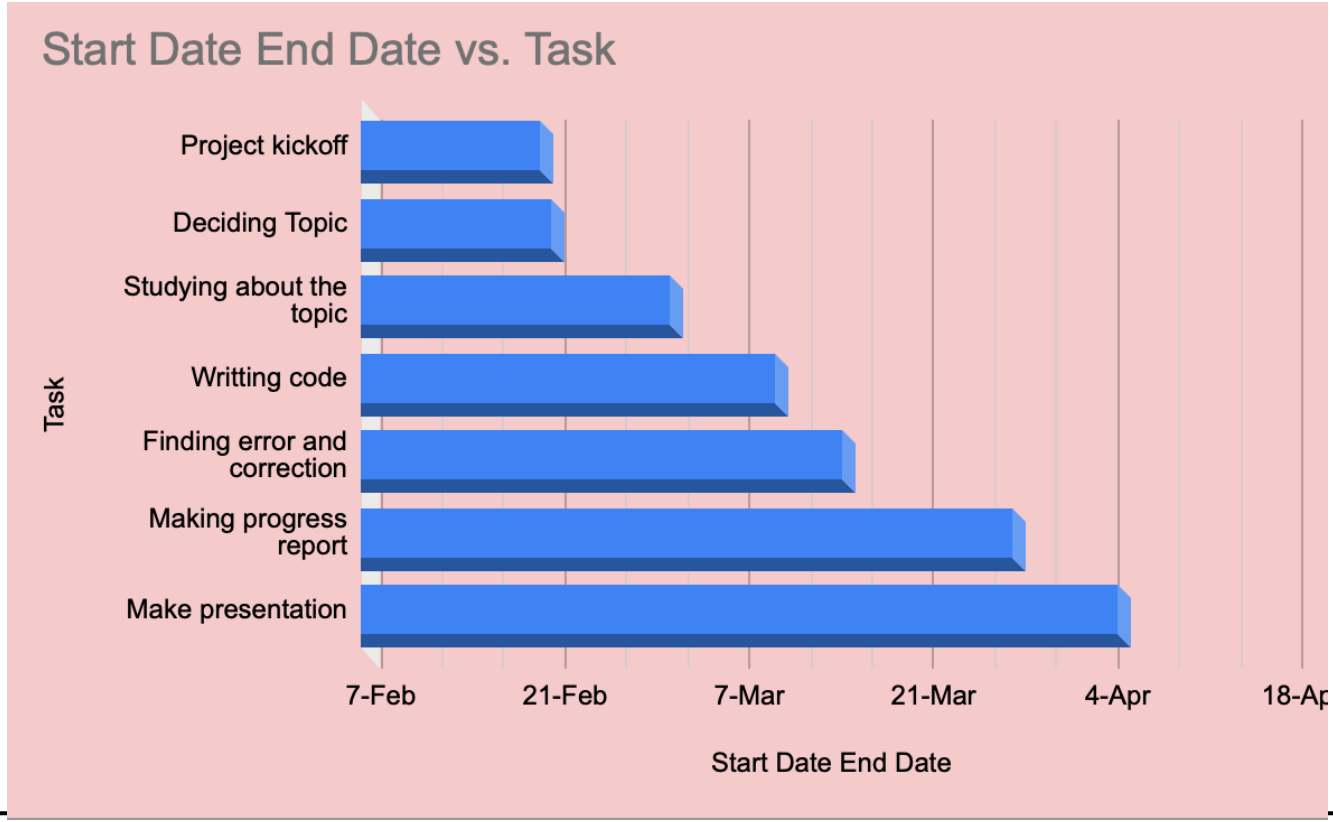
CONCLUSION :

Our sudoku solver and validator turned out to be quite fast and quite easy to use as well. There is no hiding that there are already multiple implications of Sudoku solvers available online, however, boolean implementation of this makes it quite different. Boolean makes everything easy as the outcome is always true or false and no other complicated answer, even used by computers as well. The implemented strategy is enough to find the solution to many Sudoku problems and even validate them.

FUTURE WORK:

1. Implementing a sudoku generator which will generate sudoku puzzles.
2. Making a GUI for the sudoku solver and validator.
3. Improving the time complexity of the solver.
4. Making a scoring system based on time and accuracy, and a database to keep track of the top ten records.

Progress report



REFERENCES USED:

1.https://www.researchgate.net/publication/258652094_Solving_Sudoku_using_Boolean_Algebra

2. https://ieeexplore.ieee.org/document/4804943

3. https://www.sudokuwiki.org/sudoku.htm

4. https://www.sudoku-solutions.com/

5 . https://www.researchgate.net/publication/283483103_On_the_generation_and_evaluation_of_a_complete_sudoku_puzzle


THANK YOU