SUDOKU ARRANGEMENT

A MINI PROJECT REPORT

18CSC204J -Design and Analysis of Algorithms Laboratory

Submitted by

MD FAHAD IMAM [RA2011028010122]

SYED ADEEL AHMED [RA2011028010148]

Under the guidance of

Dr. B Yamini

Assistant Professor, Department of Networking and Communication

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in Cloud Computing

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

June 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this mini project report titled “Sudoku Arrangement” is the bonafide work done by MD FAHAD IMAM (RA2011028010122) and Syed Adeel Ahmed (RA2011028010148) who carried out the mini project work and Laboratory exercises under my supervision for **18CSC204J -Design and Analysis of Algorithms Laboratory**. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr. B Yamini
ASSISTANT PROFESSOR
18CSC204J -Design and Analysis of Algorithms
Course Faculty
Department of Networking and Communications

Signature of the Internal Examiner-I **Signature of the Internal Examiner-II**

ABSTRACT

A **mathematical game** is a game whose rules, strategies, and outcomes are defined by clear mathematical parameters. Often, such games have simple rules and match procedures, such as Tic Tac Toe and Dots and Boxes. Generally, mathematical games need not be conceptually intricate to involvedeeper computational underpinnings. Mathematical games differ sharply from mathematical puzzles in that mathematical puzzles requirespecific mathematical expertise to complete, whereas mathematical games do not require a deep knowledge of mathematics to play. Often, the arithmetic core of mathematical games is not readily apparent to players untrained to note the statistical or mathematical aspects. Some mathematical games are of deep interest in the field of recreational mathematics. Sudoku is one of them. The puzzle of sudoku is one consisting of a nine-by-nine grid of squares and the grid is divided up into three-by-three grids of three-by-three squares. Each row, column and sub-grid can only contain one instance of the digits 1 through 9 and some entries are given. A correct puzzle has enough entries to ensurea unique solution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATION

SYMBOLS/ ABBREVIATION	MEANING / EXPANSION
NLU	Natural Language understanding

CHAPTER-1

PROBLEM DEFINITION

To fill the empty spaces of the given sudoku with numbers ranging between 0-9. (Boundingfunction: No same number in one row, column and block as well)

This is problem that is very appropriate for backtracking, as a digit in the wrong location oftenquickly shows that the solution is infeasible.

With *n* entries left,

- if there are no entries, *n* = 0, left, we are finished, indicate success.
- otherwise, find a square that is not yet filled, and
- for each digit from 1 to 9,
    - place the digit in the digit in that square and see whether the solution is feasible,and if so, call backtracking algorithm recursively, were
        - if the algorithm indicates success, we are finished,
        - otherwise, try the next digit; and
- if no digit works, there is no solution.

This algorithm is implemented in the source directory.

CHAPTER-2

PROBLEM EXPLANATION

To fill the empty spaces of the given sudoku with numbers ranging between 0-9.
(Bounding function: Nosame number in one row, column and block as well)
Sudoku is a number puzzle game in which numbers from 1–9 is to be placed in a 9x9 grid such that nonumbers are repeated in each row, column, and 3x3 sub grid/blocks.

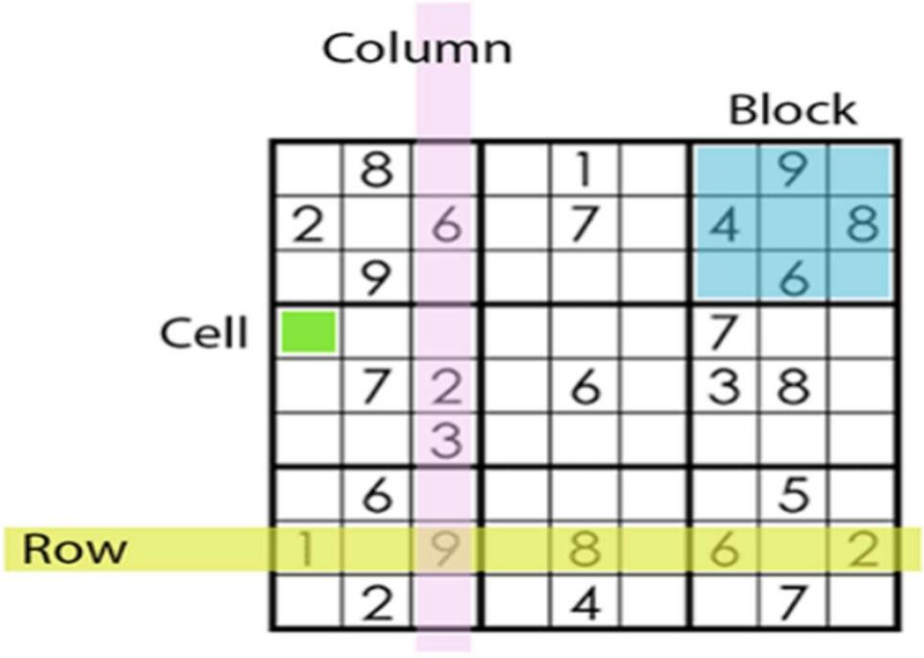

Figure 2.1

CHAPTER-3

DESIGN TECHNIQUES

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing
those solutions that fail to satisfy the constraints of the problem at any point in time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

So basically, the idea behind backtracking technique is that it searchesfora solution to a problem among all available options. Initially, we start the
backtracking from one possible option and if the problem is solved with that then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence, we understand that backtracking won't give any solution to that problem. We can also say that backtracking is a form of recursion. This is because the process of finding the solution from the various option available is repeated recursively until wedon't find the solution or e reach the final state.
So, we can conclude that backtracking at every step eliminates those choices that cannot give us the solution and proceeds to those choices that have the potential of taking us to the solution.

There are three types of problems in backtracking –

1. Decision Problem – In this, we search for a feasible solution.

2. Optimization Problem – In this, we search for the best solution.

3. Enumeration Problem – In this, we find all feasible solutions

CHAPTER-4 ALGORITHM

FOR THE PROBLEM

Create a function that checks if the given matrix is valid sudoku or not. Keep HashMap for therow, 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.

6	5		8	7	3		9	
		3	2	5				8
9	8		1		4	3	5	7
1		5						
4								2
						5		3
5	7	8	3		1		2	6
2				4	8	9		
	9		6	2	5		8	1

Figure 4.1

CHAPTER-5

EXPLANATION OF ALGORITHM

In backtracking, we first start with a sub-solution and if this sub-solution doesn't give us a correct final answer, then we just come back and change our sub solution. We are going to solve our Sudoku in a similar way. The steps which we will follow are: • If there are no unallocated cells, then the Sudoku is already solved. We will just return true. • Or else, we will fill an unallocated cell with a digit between 1 to 9 so that there are no conflicts in any of the rows, columns, or the 3x3 sub-matrices. • Now, we will try to fill the next unallocated cell and if this happens successfully, then we will return true. • Else, we will come back and change the digit we used to fill the cell. If there is no digit which fulfils the need, then we will just return false as there is no solution of this Sudoku

6	5	1	8	7	3	2	9	4
7	4	3	2	5	9	1	6	8
9	8	2	1	6	4	3	5	7
1	2	5	4	3	6	8	7	9
4	3	9	5	8	7	6	1	2
8	6	7	9	1	2	5	4	3
5	7	8	3	9	1	4	2	6
2	1	6	7	4	8	9	3	5
3	9	4	6	2	5	7	8	1

Figure 4.2

CHAPTER-6

COMPLEXITY ANALYSIS

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

CHAPTER-7

CONCLUSION

Thus, we have successfully found the solution for solving sudoku using backtracking withminimal time and space required.

REFERENCES

1. https://www.cs.rochester.edu/u/brown/242/assts/termprojs/Sudoku09.pdf
2. https://en.wikipedia.org/wiki/Mathematical_game
3. https://ece.uwaterloo.ca/~dwharder/aads/Algorithms/Backtracking/Sudoku/#1
4. https://www.geeksforgeeks.org/sudoku-backtracking-7/
5. https://www.programiz.com/dsa/backtracking- algorithm#:~:text=A%20backtracking%20algorithm%20is%20a,chooses%20the%20desired%2Fbest%20solutions.

APPENDIX

CODE

```
#include <stdio.h>

#define SIZE 9

//sudoku problem
int matrix[9][9] = {
   {6,5,0,8,7,3,0,9,0},
   {0,0,3,2,5,0,0,0,8},
   {9,8,0,1,0,4,3,5,7},
   {1,0,5,0,0,0,0,0,0},
   {4,0,0,0,0,0,0,0,2},
   {0,0,0,0,0,0,5,0,3},
   {5,7,8,3,0,1,0,2,6},
   {2,0,0,0,4,8,9,0,0},
   {0,9,0,6,2,5,0,8,1}
};

//function to print sudoku
void print_sudoku()
{
   int i,j;
   for(i=0;i<SIZE;i++)
   {
      for(j=0;j<SIZE;j++)
      {
         printf("%d\t",matrix[i][j]);
      }
      printf("\n\n");
   }
}

//function to check if all cells are assigned or not
//if there is any unassigned cell
//then this function will change the values of
//row and col accordingly
int number_unassigned(int *row, int *col)
{
   int num_unassign = 0;
```

```
    int i,j;
    for(i=0;i<SIZE;i++)
    {
        for(j=0;j<SIZE;j++)
        {
            //cell is unassigned
            if(matrix[i][j] == 0)
            {
                //changing the values of row and col
                *row = i;
                *col = j;
                //there is one or more unassigned cells
                num_unassign = 1;
                return num_unassign;
            }
        }
    }
    return num_unassign;
}

//function to check if we can put a
//value in a paticular cell or not
int is_safe(int n, int r, int c)
{
    int i,j;
    //checking in row
    for(i=0;i<SIZE;i++)
    {
        //there is a cell with same value
        if(matrix[r][i] == n)
            return 0;
    }
    //checking column
    for(i=0;i<SIZE;i++)
    {
        //there is a cell with the value equal to i
        if(matrix[i][c] == n)
            return 0;
    }
    //checking sub matrix
    int row_start = (r/3)*3;
    int col_start = (c/3)*3;
    for(i=row_start;i<row_start+3;i++)
```

```
    {
        for(j=col_start;j<col_start+3;j++)
        {
            if(matrix[i][j]==n)
                return 0;
        }
    }
    return 1;
}

//function to solve sudoku
//using backtracking
int solve_sudoku()
{
    int row;
    int col;
    //if all cells are assigned then the sudoku is already solved
    //pass by reference because number_unassigned will change the values of row and col
    if(number_unassigned(&row, &col) == 0)
        return 1;
    int n,i;
    //number between 1 to 9
    for(i=1;i<=SIZE;i++)
    {
        //if we can assign i to the cell or not
        //the cell is matrix[row][col]
        if(is_safe(i, row, col))
        {
            matrix[row][col] = i;
            //backtracking
            if(solve_sudoku())
                return 1;
            //if we can't proceed with this solution
            //reassign the cell
            matrix[row][col]=0;
        }
    }
    return 0;
}

int main()
{
    if (solve_sudoku())
```

```
    print_sudoku();
else
    printf("No solution\n");
return 0;
  }
```

CODE SCREENSHOTS

```c
#include <stdio.h>

#define SIZE 9

//sudoku problem
int matrix[9][9] = {
    {6,5,0,8,7,3,0,9,0},
    {0,0,3,2,5,0,0,0,8},
    {9,8,0,1,0,4,3,5,7},
    {1,0,5,0,0,0,0,0,0},
    {4,0,0,0,0,0,0,0,2},
    {0,0,0,0,0,0,5,0,3},
    {5,7,8,3,0,1,0,2,6},
    {2,0,0,0,4,8,9,0,0},
    {0,9,0,6,2,5,0,8,1}
};

//function to print sudoku
void print_sudoku()
{
    int i,j;
    for(i=0;i<SIZE;i++)
    {
        for(j=0;j<SIZE;j++)
        {
            printf("%d\t",matrix[i][j]);
        }
        printf("\n\n");
    }
}

//function to check if all cells are assigned or not
//if there is any unassigned cell
//then this function will change the values of
//row and col accordingly
int number_unassigned(int *row, int *col)
{
    int num_unassign = 0;
    int i,j;
    for(i=0;i<SIZE;i++)
    {
        for(j=0;j<SIZE;j++)
        {
            //cell is unassigned
            if(matrix[i][j] == 0)
            {
                //changing the values of row and col
                *row = i;
```

```
                *col = j;
                //there is one or more unassigned cells
                num_unassign = 1;
                return num_unassign;
            }
        }
    }
    return num_unassign;
}

//function to check if we can put a
//value in a paticular cell or not
int is_safe(int n, int r, int c)
{
    int i,j;
    //checking in row
    for(i=0;i<SIZE;i++)
    {
        //there is a cell with same value
        if(matrix[r][i] == n)
            return 0;
    }
    //checking column
    for(i=0;i<SIZE;i++)
    {
        //there is a cell with the value equal to i
        if(matrix[i][c] == n)
            return 0;
    }
    //checking sub matrix
    int row_start = (r/3)*3;
    int col_start = (c/3)*3;
    for(i=row_start;i<row_start+3;i++)
    {
        for(j=col_start;j<col_start+3;j++)
        {
            if(matrix[i][j]==n)
                return 0;
        }
    }
    return 1;
}

//function to solve sudoku
//using backtracking
int solve_sudoku()
{
    int row;
    int col;
    //if all cells are assigned then the sudoku is already solved
    //pass by reference because number_unassigned will change the values of row and col
    if(number_unassigned(&row, &col) == 0)
        return 1;
    int n,i;
```

```
    //number between 1 to 9
    for(i=1;i<=SIZE;i++)
    {
        //if we can assign i to the cell or not
        //the cell is matrix[row][col]
        if(is_safe(i, row, col))
        {
            matrix[row][col] = i;
            //backtracking
            if(solve_sudoku())
                return 1;
            //if we can't proceed with this solution
            //reassign the cell
            matrix[row][col]=0;
        }
    }
    return 0;
}

int main()
{
    if (solve_sudoku())
        print_sudoku();
    else
        printf("No solution\n");
    return 0;
}
```