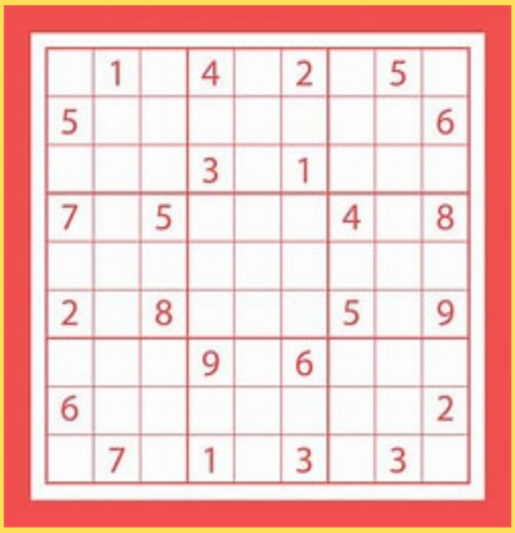
DAA MINI PROJECT REPORT

TOPIC - SUDOKU USING BACKTRACKING

SUBMISSION TO- MRS.LAVANYA V



SUBMISSION BY-

NYASA GUPTA (RA2011031010068)
DHRUMIT SHAH(RA2011031010075)
PRATEEK JAIN(RA2011031010082)
UTKARSH SINGH(RA2011031010090)

COMPUTER SCIENCE ENGINEERING

with specialization in INFORMATION TECHNOLOGY

DEPARTMENT OF NETWORKING AND COMMUNICATIONSCOLLEGE OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SUDOKU USING BACKTRACKING

Submitted By

NYASA GUPTA (RA2011031010068)

DHRUMIT SHAH (RA2011031010075)

PRATEEK JAIN(RA2011031010082)

UTKARSH SINGH (RA2011031010090)

Under the Guidance of

Mrs. Lavanya V

(Assistant Professor, Dept of Networking and Communications)

SRMI INSTITUTE OF SCIENCE AND TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR -603 203

BONAFIDECERTIFICATE

Register No.________________

Certified to be bonafide record of the work done by ____Nyasa Gupta, Dhrumit Shah, Prateek Jain , Utkarsh Singh_______ *of* ____CSE-IT B.tech degree course in the practical 18CSC204J-Design and Analysis of Algorithms in SRM Institute of Science and Technology, Kattankulathur during the academic year 2021-22.

Date: **Lab Incharge:**

Submitted for university examination held in___________________

SRMInstitute of Science and Technology, Kattankulathur.

CONTENTS

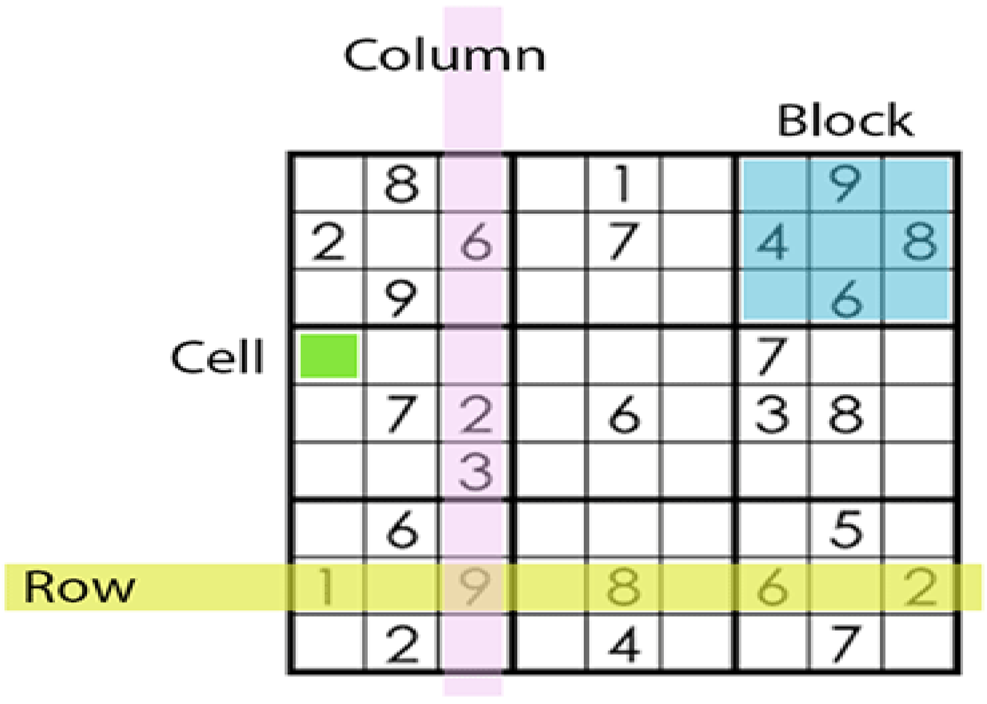
PROBLEM STATEMENT

To fill the empty spaces of the given sudoku with numbers ranging between 0-9.

(Bounding function : No same number in one row ,column and block as well)

SUDOKU

Sudoku is a number puzzle game in which numbers from 1–9 are to be placed in a 9x9 grid such that no numbers are repeated in each row, column, and 3x3 sub-grid/blocks.



Solutions

Approach used – <u>Backtracking</u>

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.

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

Basic Approach

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:

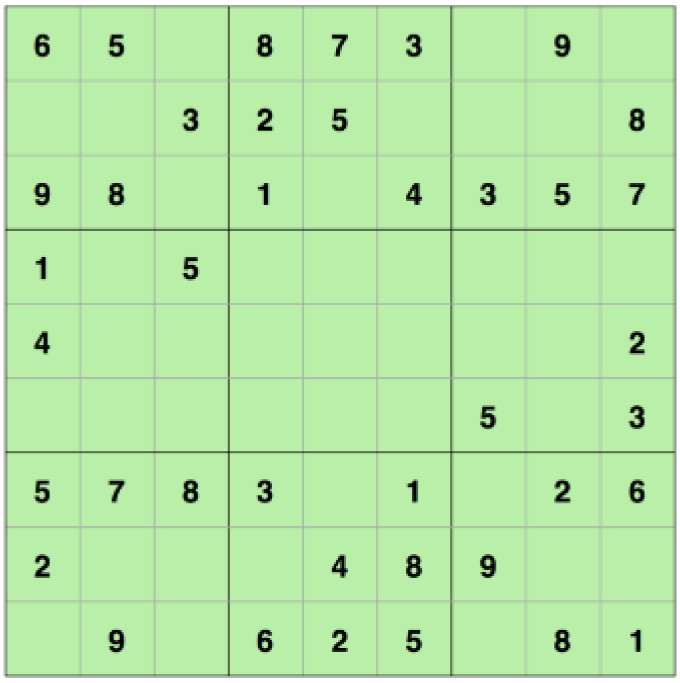
1) Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true;
2) Create a recursive function that takes a grid and the current row and column index.
3) 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.

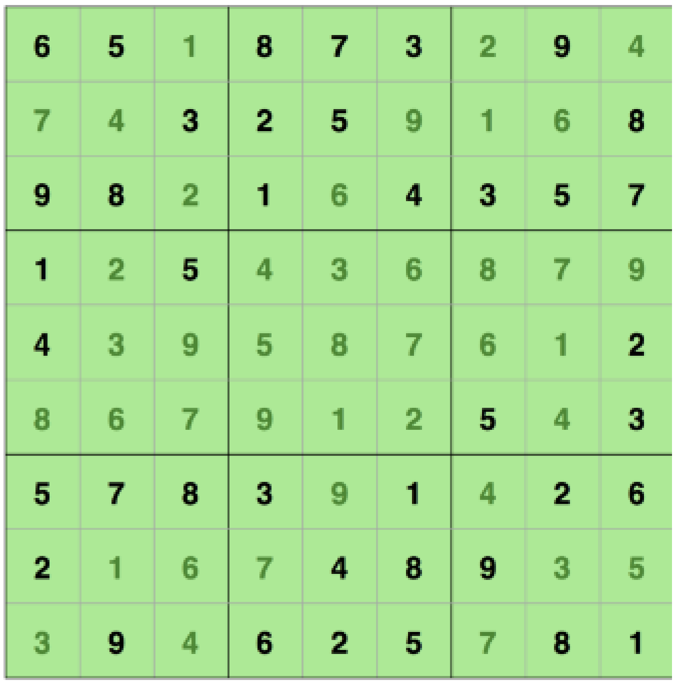
Sample Input / Output

Input:



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

Output:

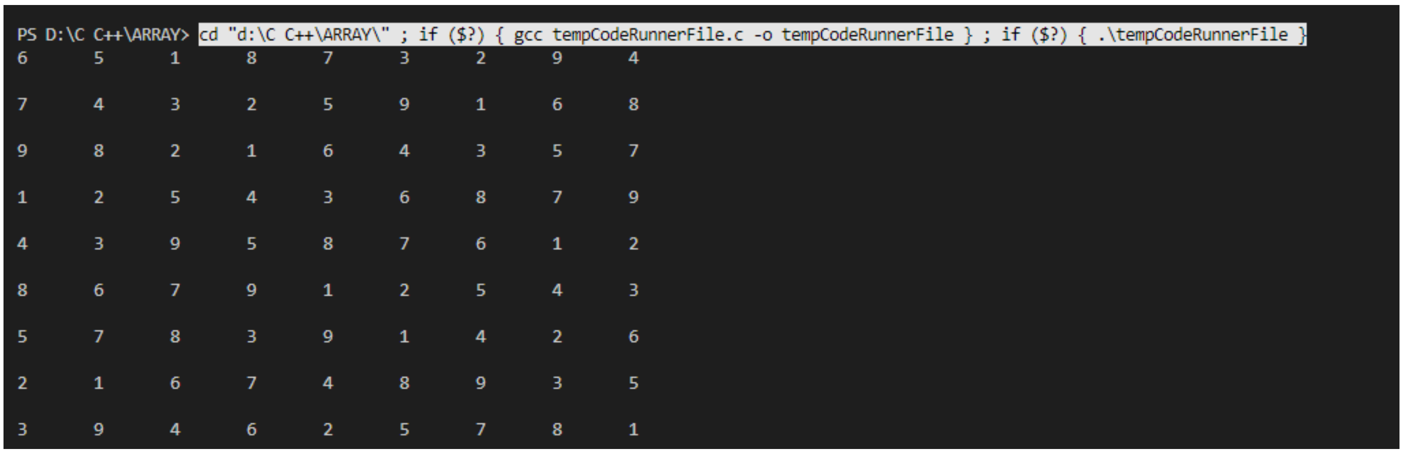


6	5	1	8	7	3	2	9	4
7	4	3	2	5	9	1	6	8
9	8	2	1	6	4	3	5	7
1	2	5	4	3	6	8	7	9
4	3	9	5	8	7	6	1	2
8	6	7	9	1	2	5	4	3
5	7	8	3	9	1	4	2	6
2	1	6	7	4	8	9	3	5
3	9	4	6	2	5	7	8	1

Code implementation

```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;
}
```

Output:



```
PS D:\C C++\ARRAY> cd "d:\C C++\ARRAY\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
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
```

Explanation of code

Initially, we are just making a matrix for Sudoku and filling its unallocated cells with 0. Thus, the matrix contains the Sudoku problem and the cells with value 0 are vacant cells.

print_sudoku() → This is just a function to print the matrix.

number_unassigned → This function finds a vacant cell and makes the variables ‘row’ and ‘col’ equal to indices of that cell. In C, we have used pointers to change the value of the variables (row, col) passed to this function (pass by reference). In Java and Python, we are returning an array (or list, for Python) which contains these values. So, this function tells us if there is any unallocated cell or not. And if there is any unallocated cell then this function also tells us the indices of that cell.

is_safe(int n, int r, int c) → This function checks if we can put the value ‘n’ in the cell (r, c) or not. We are doing so by first checking if there is any cell in the row ‘r’ with the value ‘n’ or not – if(matrix[r][i] == n). Then we are checking if there is any cell with the value ‘n’ in the column ‘c’ or not – if(matrix[i][c] == n). And finally, we are checking for the sub-matrix. (r/3)*3 gives us the starting index of the row r. For example, if the value of ‘r’ is 2 then it is in the sub-matrix which starts from (0, 0). Similarly, we are getting the value of starting column by (c/3)*3. Thus, if a cell is (2,2), then this cell will be in a sub-matrix which starts from (0,0) and we are getting this value by (c/3)*3 and (r/3)*3. After getting the starting indices, we can easily iterate over the sub-matrix to check if the we can put the value ‘n’ in that sub-matrix or not.

solve_sudoku() → This is the actual function which solves the Sudoku and uses backtracking. We are first checking if there is any unassigned cell or not by using the number_unassigned function and if there is no unassigned cell then the Sudoku is solved. number_unassigned function also gives us the indices of the vacant cell. Thus, if there is any vacant cell then we will try to fill this cell with a value between 1 to 9. And we will use the is_safe to check if we can fill a particular value in that cell or not. After finding a value, we will try to solve the rest of the Sudoku solve_sudoku. If this value fails to solve the rest, we will come back and try another value for this cell matrix[row][col]=0;. The loop will try other values in the cell.

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)). 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.

Conclusion

Successfully , found the solution for solving sudoku using backtracking with minimal time and space required. The term recursive backtracking comes from the way in which the problem tree is explored. The algorithm tries a value, then optimistically recurs on the next cell and checks if the solution (as built up so far) is valid. If, at some later point in execution, it is determined that what was built so far will not solve the problem, the algorithm backs up the stack to a state in which there could be another valid solution.

Other ways to solve sudoku are:

- Crook's algorithm
- Constraint programming
- AI based approach

Reference

- Sudoku | Backtracking-7 - GeeksforGeeks
- Backtracking Introduction - javatpoint