(class 5) 26-03-2023

Backtracking

Date..................

Ques Sudoku Solver.

Given a 9x9 matrix, i.e.,

	I			II			III	
	IV			V			VI	
	VII			VIII			IX	

constraints :-

i) In each column, there should be numbers from 1 to 9 & should not be repeating.

ii) In each row, there should be numbers from 1 to 9 & should not be repeating.

iii) There will be 9, 3x3 boxes & each box should have numbers from 1 to 9 & there should be no repetition.

For example, if we are trying to insert 2 in the empty cell, then 2 should not be present in the current row, current column & in current 3x3 box.

The puzzle must have at least 17 clues to have a valid solutions. There can be more than 17 solutions / clues but minimum 17 clues should be present.

It means, we've given a partially filled 9x9 2D matrix, & the goal is to assign digits from 1 to 9 to the empty cells, so that every row column & subgrid of size 3 x 3 contains exactly one instance of the digits from 1 to 9.

	0	1	2	3	4	5	6	7	8
0	3		6	5		8	4		
1	5	2							
2		8	7					3	1
3			3		1			8	
4	9			8	6	3			5
5		5			9		6		
6	1	3					2	5	
7								7	4
8			5	2		6	3		

the already present values are called "clues".

Date..................

If there are many clues, then less chances of multiple solutions, & if there are less clues, then there is high chances of multiple solutions. In our given matrix, we've to fill the empty cells only, we can't changes clues.

So, we will start from cell 0,1 & apply isSafe function on that cell. isSafe function will check that the number we are trying to place in the current cell is present in the same row, row should not contain the same number, isSafe also check the, the number should not present in the same column, also the number should not present in the current 3 x 3 box.

If the number is not present, then isSafe return true, else isSafe return false.

If true, then we insert the number at that cell & move ahead. If false, then we try to insert another number. suppose, we are not able to insert any number from 1 to 9 at that cell then this means that there is a fault in previous placement, just like n-queen problem.

This is the only logic, we need to apply until we fill all the cells.

code:-

```
#include<bits/ stdc++.h>
using namespace std;
//function to print sudoku
void printsudoku (int board[9][9], int n)
{
    cout <<endl;
    for(int i=0; i<n; i++){
        for(int j = 0; j< n; j++){
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
}
//this function returns true if placing
 the current value at given cell is
safe or not, if safe then true else false.
// we have 3 conditions to check
//1. check the given value in the same
        row.
//2. check the given value in the same
        column.
//3. check the given value in the
        given 3x3 sub-matrix.
bool isSafe(int value, int board[9][9],
                                int i, int j)
{
    // checking row, i represent row
    // for checking row, i will remain
        same & j will move.
    // j represent column
for(int column = 0; column < 9;
                              column++)
{
    if(board[i][column] == value)
    {
        return false;
    }
}
// checking column, j represent
                              column.
// for checking column, j will rema
   -in same & i will move
// i represent row.
for(int row = 0; row < 9; row++)
{
    if (board[row][j] == value)
    {
        return false;
    }
}
// now check 3x3 sub-matrix.
for(int R = 0; R < 9; R++)
{
    if(board[3 * (i / 3) + (R / 3)][3 *
        (j / 3) + (R % 3)] == value)
    {
        return false;
    }
}
// if false is not returned, then true
return true;
}
```

```
//this function return true or false
basis on we found a solution or not
bool solvesudoku(int board[9][9],
                              int n)
{
    //using nested loops we are traversing
    on each & every value of the given
    matrix.
    for(int i = 0; i< n; i++){
      for(int j = 0; j< n; j++){
        //we're checking for empty cell
        // if the cell is empty then only
        we can insert value.
        // if cell is not empty, it means
        its a clue & we can't change
        if(board[i][j] == 0){
          for(int value =1; value <=9;
                              value ++){
            if(issafe(value, board, i, j)
            {
              // if inserting a value is
              safe then only we place
              board[i][j] = value;
              //recursive call
              bool nextvalue = solvesudoku(board, n);
              // if we successfully
              solve the next value
              then we don't need backtracking, & because
              of this we don't need
              backtracking
```

```
                    if(nextValue ==true){
                        return true;
                    }
                    else{
                        board[i][j] = 0;
                    }
                }
            }
            // if no value from 1 to 9 can
            insert then return false.
            return false;
        }
      }
    }
    return true; // if all cells are filled
    with error then return true.
}
int main()
{
    int n = 9;
    // 0 represents empty cells
    int board[9][9] = {
                        {2,0,9,0,0,0,6,0,0},
                        {0,4,0,8,7,0,0,1,2},
                        {8,0,0,0,1,9,0,4,0},
                        {0,3,0,7,0,0,8,0,1},
                        {0,6,5,0,0,8,0,3,0},
                        {1,0,0,0,3,0,0,0,7},
                        {0,0,0,6,5,0,7,0,9},
                        {6,0,4,0,0,0,0,2,0},
                        {0,8,0,3,0,1,4,5,0}};
    cout << "Sudoku before solve:";
    printSudoku(board, n);
    if(solveSudoku(board, n))
    {
        cout<<"Sudoku after solve:";
        printSudoku(board, n);
    }
    else
    {
        cout << "In NO solution";
    }
    cout << endl;
    return 0;
}
```

- understanding 3 x 3 box check condition:-

sent in safe

i) i = 0 to 8

board[3 * (i/3) + (k/3)][3 * (j/3) + (k%3)]

	0	1	2
0	4	5	
1			2
2			

→ (i, j) → (0, 2)

```
* R = 0,
  board[3 * (0/3) + (0/3)][3 * (2/3) +
                               (0 % 3)]

  board[3 * 0 + 0][3 * 0 + 0]
  board[0][0]
* R = 1,
  board[3 * (0/3) + (1/3)][3 * (2/3) +
                               (1 % 3)]

  board[3 * 0 + 0][3 * 0 + 1]
  board[0][1]
* R = 2,
  board[3 * (0/3) + (2/3)][3 * (2/3) +
                               (2 % 3)]

  board[3 * 0 + 0][3 * 0 + 2]
  board[0][2]
* R = 3,
  board[3 * (0/3) + (3/3)][3 * (2/3) +
                               (3 % 3)]

  board[3 * 0 + 1][3 * 0 + 0]
  board[1][0]
* R = 4,
  board[3 * (0/3) + (4/3)][3 * (2/3) +
                               (4 % 3)]

  board[3 * 0 + 1][3 * 0 + 1]
  board[1][1]
* R = 5,
  board[3 * (0/3) + (5/3)][3 * (2/3) +
                               (5 % 3)]

  board[3 * 0 + 1][3 * 0 + 2]
  board[1][2]
```

* R = 6,
board[3 * (0/3) + (6/3)][3 * (2/3) + (6 % 3)]
board[3 * 0 + 2][3 * 0 + 0]
board[2][0]

* R = 7,
board[3 * (0/3) + (7/3)][3 * (2/3) + (7 % 3)]
board[3 * 0 + 2][3 * 0 + 1]
board[0 + 2][0 + 1]
board[2][1]

* R = 8,
board[3 * (0/3) + (8/3)][3 * (2/3) + (8 % 3)]
board[3 * 0 + 2][3 * 0 + 2]
board[2][2]

using the above formula, we can check the 3 x 3 matrix & check the value,

Note :-

$$\underbrace{3 * (i/3)}_{\text{starting row of each box of 3 x 3}} + \underbrace{(R + 3)}_{\text{movement in down direction.}}$$

$$\underbrace{3 * (j/3)}_{\text{starting column of each box of } 3 \times 3} + \underbrace{(k \% 3)}_{\text{movement in right direction}}$$