Solving Sudoku problem using Constraint Satisfaction Search

Under guidance of Prof. Punam Bedi

Important Components of Constraint Satisfaction Problem (CSP)

1.Variable
2.Domain
3.Constraints
4.Worlds
5.Model

In our sudoku, define them :

1.Every empty cell is a **variable**.

2.Each variable can take one of 1 to N numbers each. This is the **domain** for the variable.

3.Each column, each row, and each of the nine N x N sub-grids (also called boxes) contains one of all of the digits 1 through N.

4.One possible combination of values is called a "**world**". So there are (d^n) worlds. d is the size of the domain and n is the number of variables.

5.A **model** is a world which meets all the constraints.

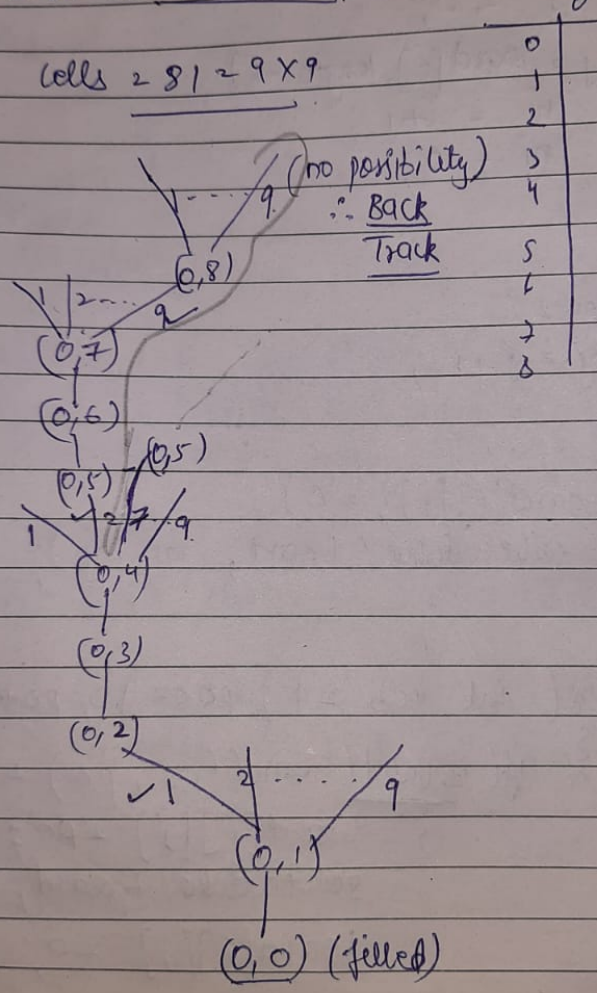
Algorithm using Backtracking:

Page No.
Date

Sudoku

→ We will use recursion.

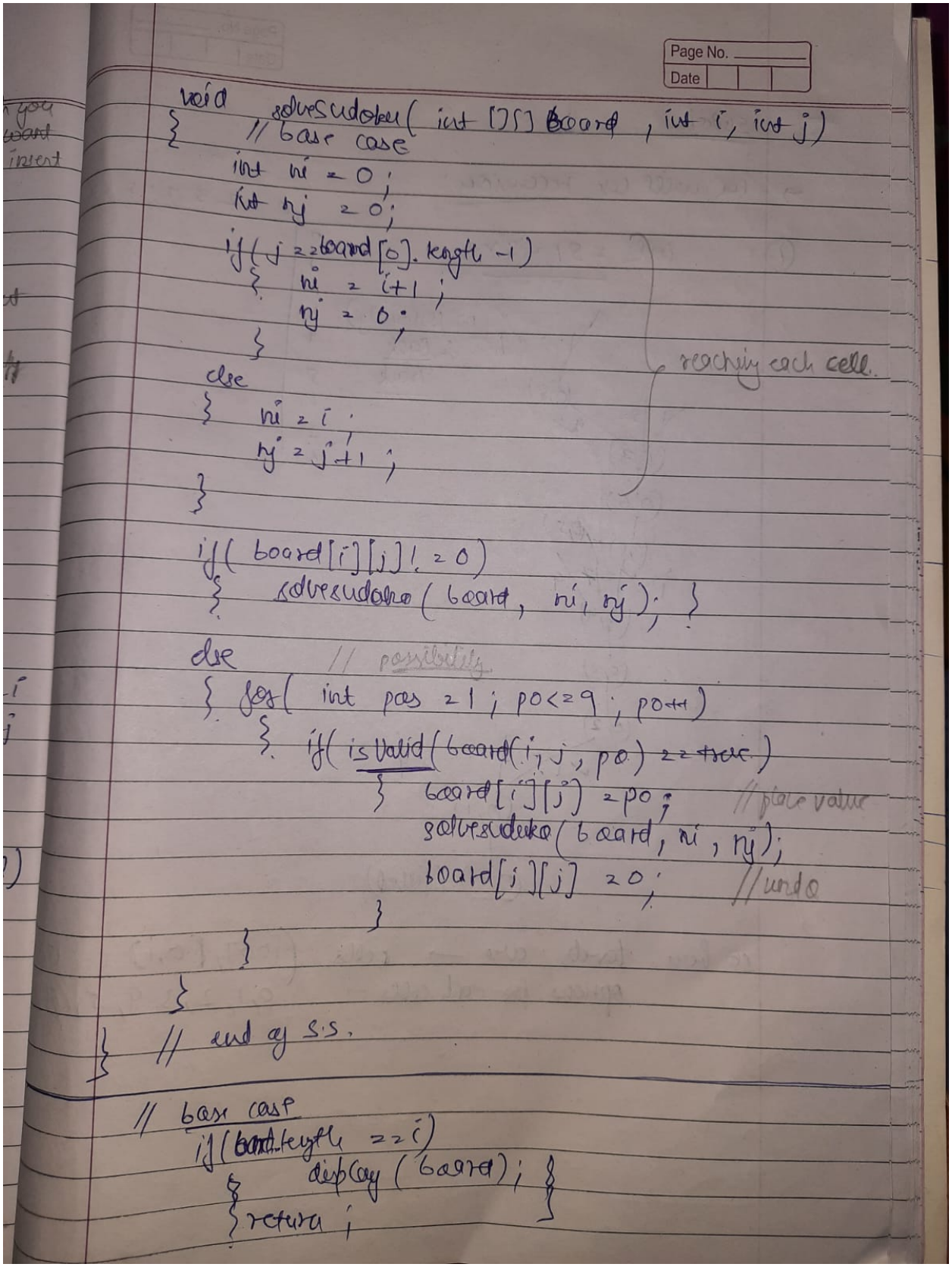
① cells = 81 = 9 × 9



0 1 2 3 4 5 6 7 8

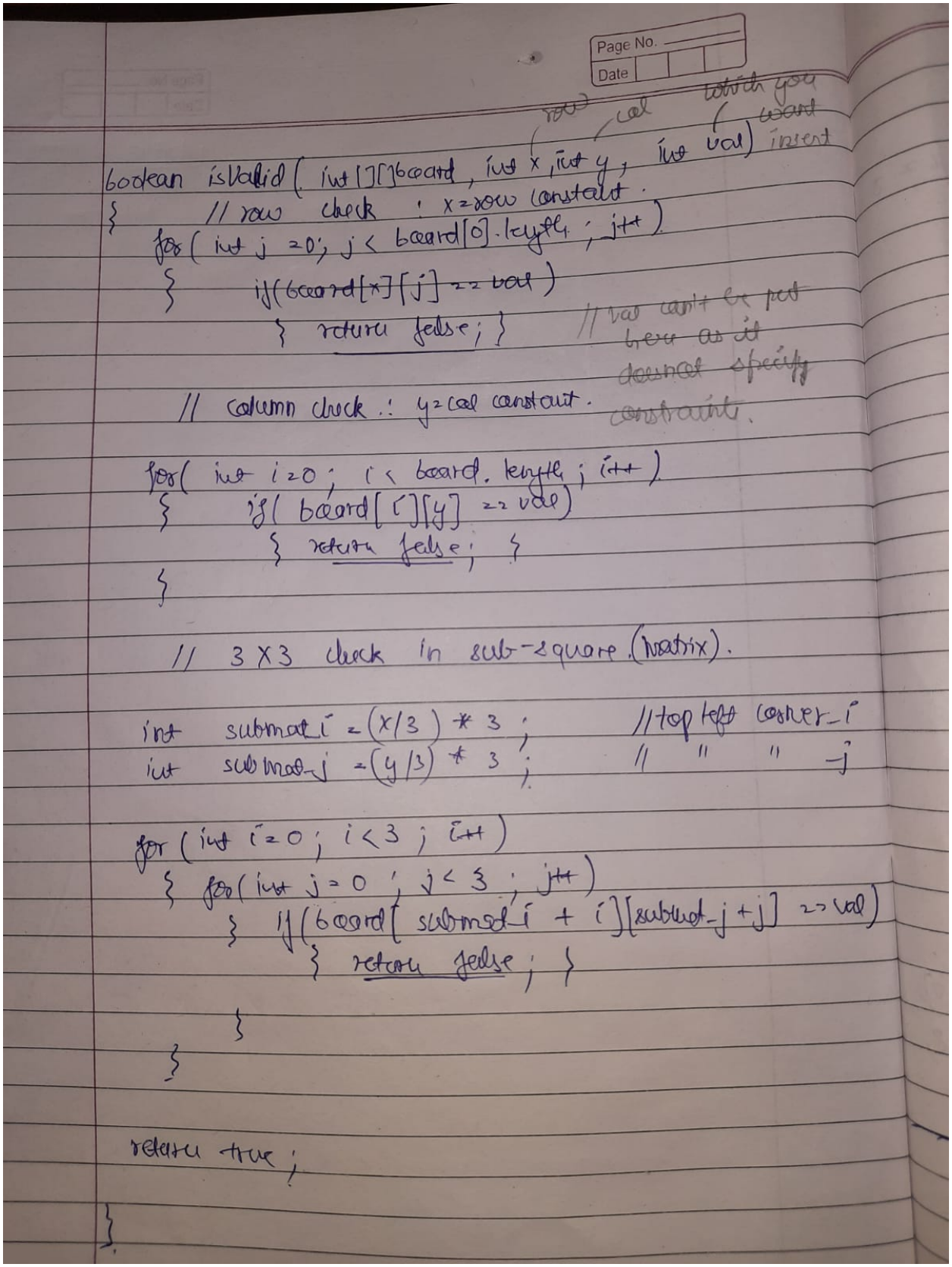
0
1
2
3
4
5
6
7
8

so here levels are → cells ([00], [0,1] .... [8,8])
options for each cells → 0, 1, 2, 3, 4, 5, 6, 7, 8,

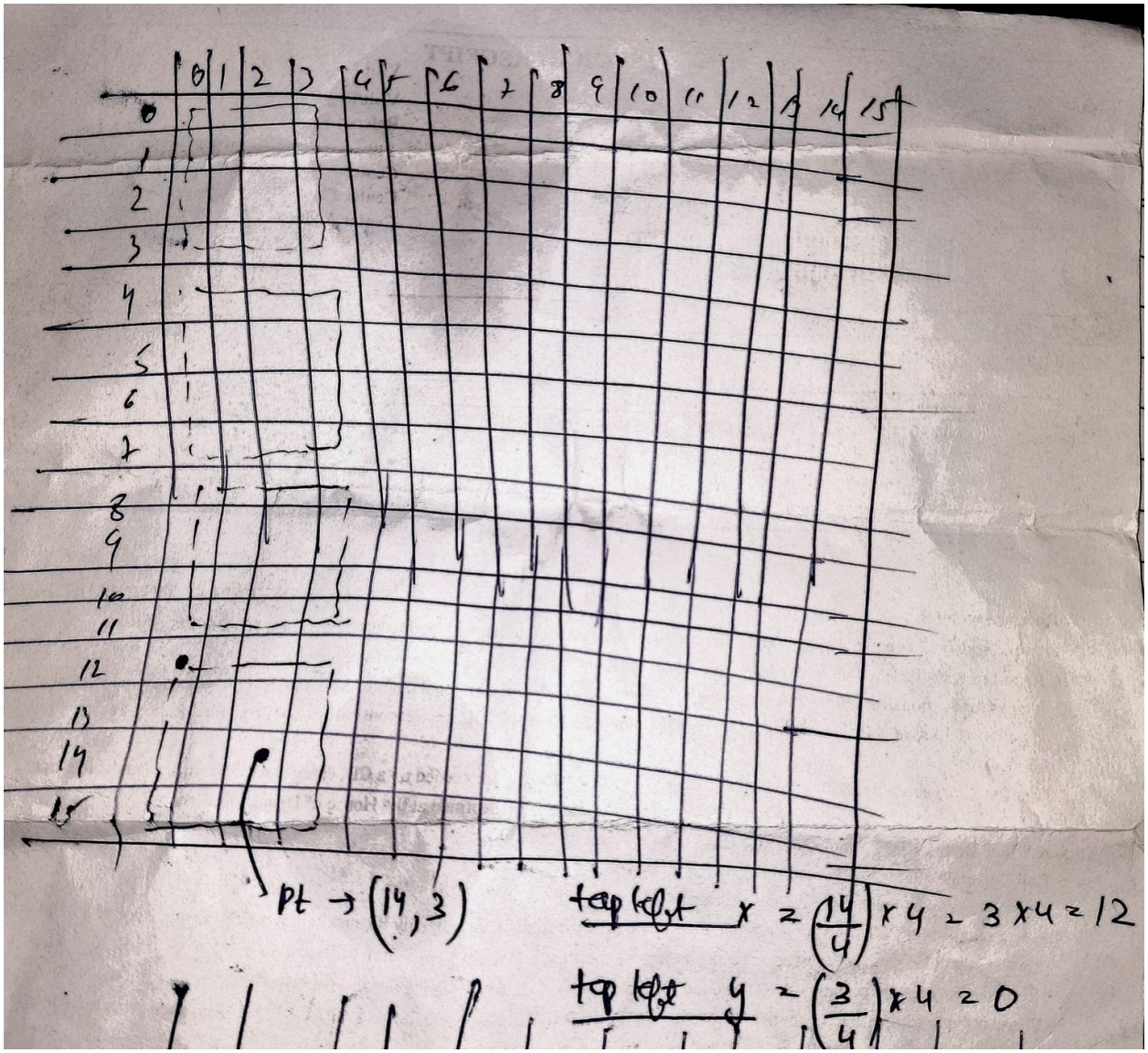


```
void solveSudoku( int [][] Board , int i, int j)
{   // base case
    int ni = 0;
    int nj = 0;
    if(j == board[0].length - 1)
    {   ni = i+1;
        nj = 0;
    }
    else
    {   ni = i;
        nj = j+1;
    }

    if( board[i][j] != 0)
    {   solveSudoku( board, ni, nj); }

    else                    // possibility
    {   for( int pos = 1; po<=9; po++)
        {   if( isValid(board(i, j, po) == true)
            {   board[i][j] = po;          //place value
                solveSudoku( board, ni, nj);
                board[i][j] = 0;           //undo
            }
        }
    }
}   // end of S.S.

// base case
    if(board.length == i)
    {   display( board); }
    return;
```

reaching each cell.

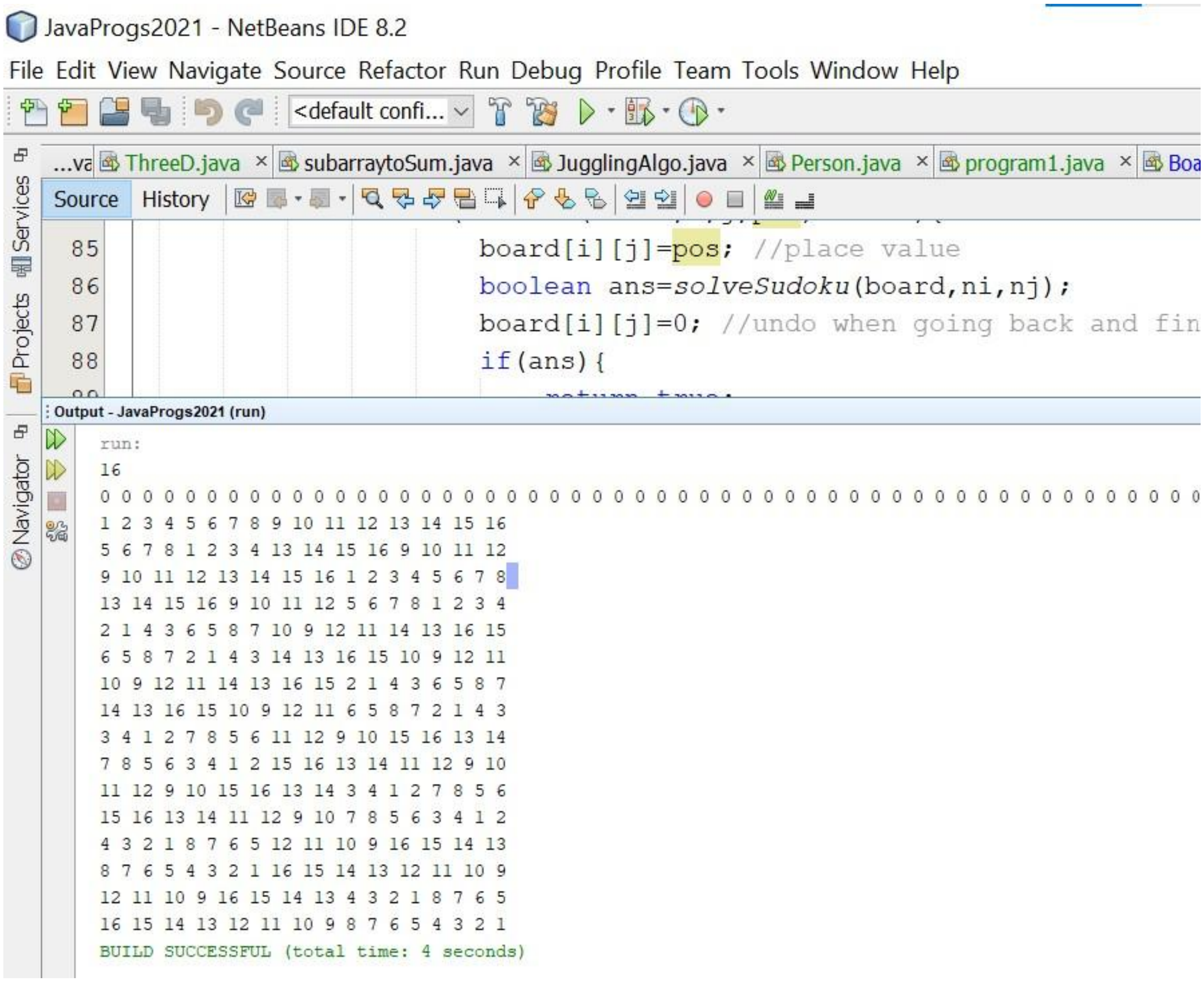


```
boolean isValid( int[][]board, int x, int y, int val)
{
    // row check : x=row constant.
    for( int j =0; j < board[0].length ; j++)
    {   if(board[x][j] == val)
        { return false; }
    }

    // column check : y=col constant.

    for( int i=0 ; i < board.length ; i++)
    {   if( board[i][y] == val)
        { return false; }
    }

    // 3 X 3 check in sub-square (matrix).

    int submat_i = (x/3) * 3 ;
    int submat_j = (y/3) * 3 ;

    for (int i=0; i<3 ; i++)
    { for(int j = 0 ; j< 3 ; j++)
        { if(board[ submat_i + i][submat_j +j] == val)
            { return false; }

        }
    }

    return true;

}
```



This image show how it will calculate sub matrix in case of 16 x 16.

Output from algorithm:


JavaProgs2021 - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default confi...
...va ThreeD.java subarraytoSum.java JugglingAlgo.java Person.java program1.java Boa
Source History
Services
Projects
Navigator
85 board[i][j]=pos; //place value
86 boolean ans=solveSudoku(board,ni,nj);
87 board[i][j]=0; //undo when going back and fin
88 if(ans){
Output - JavaProgs2021 (run)
run:
16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
5 6 7 8 1 2 3 4 13 14 15 16 9 10 11 12
9 10 11 12 13 14 15 16 1 2 3 4 5 6 7 8
13 14 15 16 9 10 11 12 5 6 7 8 1 2 3 4
2 1 4 3 6 5 8 7 10 9 12 11 14 13 16 15
6 5 8 7 2 1 4 3 14 13 16 15 10 9 12 11
10 9 12 11 14 13 16 15 2 1 4 3 6 5 8 7
14 13 16 15 10 9 12 11 6 5 8 7 2 1 4 3
3 4 1 2 7 8 5 6 11 12 9 10 15 16 13 14
7 8 5 6 3 4 1 2 15 16 13 14 11 12 9 10
11 12 9 10 15 16 13 14 3 4 1 2 7 8 5 6
15 16 13 14 11 12 9 10 7 8 5 6 3 4 1 2
4 3 2 1 8 7 6 5 12 11 10 9 16 15 14 13
8 7 6 5 4 3 2 1 16 15 14 13 12 11 10 9
12 11 10 9 16 15 14 13 4 3 2 1 8 7 6 5
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
BUILD SUCCESSFUL (total time: 4 seconds)