

DATA STRUCTURES AND ALGORITHMS – 1

BACKTRACKING : CLASSIC PROBLEMS

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

CSE(AI)



Centre for Computational Engineering and Networking

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

JULY – 2023

A THESIS

Submitted by

GROUP 14

GANESH SUNDHAR S (CB.EN.U4AIE22017)

KARTHIGAI SELVAM R (CB.EN.U4AIE22025)

SHRUTHIKAA V (CB.EN.U4AIE22047)

BHAVYA SAINATH (CB.EN.U4AIE22055)

PROBLEM STATEMENT :

The main objective of this project is to solve and implement 4 of the classic problems using backtracking algorithms.

1. **N Queens :** The problem is to print all the possible ways to place n queens in a $n \times n$ board where all the queens are in a non-attacking position.
2. **Knight Tour :** The problem is to traverse a knight in a $n \times n$ board where each square has to be traversed once and only once.
3. **Maze Challenge :** The problem is to print all the possible ways of traversing a complete rectangular board of given size where the only moves are to move up, down, right or left.
4. **Sudoku Solver :** The problem is to solve a sudoku puzzle.

Reason : We took this problem in order to get a better understanding of the array data structure and the backtracking algorithm.

METHODOLOGY :

We used the array data structure for all the problems mentioned above. We used this data structure because it is easier to implement the board of all the problems mentioned above using a 2D array.

Advantages of array :

1. Access the values with a time complexity of $O(1)$
2. Arrays are efficient when using them for storing data of fixed size as arrays have contiguous memory locations allocated.
3. Easier iteration to solve values based on the data stored.
4. Multidimensional arrays provide more functionalities for our purpose.
5. Compact representation for better learning

Disadvantages of array :

1. Size is fixed and resizing requires a lot of time.
2. Dynamic allocation is not possible.
3. Searching process has a worst case time complexity of $O(n)$
4. Wastage of memory locations if there are unused memory locations.

RESULT :

1) Nqueens :

❖ Input

```
PS F:\Ganesh\Amrita\Subjects\Sem 2\Data Structures and Algorithms\Project\Group_14_Batch_A> java Nqueens.java
Enter the size of the board : 4
```

❖ Board Size : 4

```
Enter the size of the board : 4
X Q X X
X X X Q
Q X X X
X X Q X

X X Q X
Q X X X
X X X Q
X Q X X

The total number of ways is : 2
```

❖ Board Size : 5

```
X X X X Q
X X Q X X
Q X X X X
X X X Q X
X Q X X X

X X X X Q
X Q X X X
X X Q X X
Q X X X X
X Q X X X

X X X Q X
X Q X X X
X X X X Q
X X Q X X
Q X X X X

X X X Q X
X Q X X X
Q X X X X
X X Q X X
X X X X Q
X Q X X X

X X Q X X
X X X X Q
X Q X X X
X X X Q X
X X X Q X
Q X X X X

The total number of ways is : 10
```

❖ Board Size : 8

```
X X X X X X X Q
X X Q X X X X X
Q X X X X X X X
X X X X X Q X X
X Q X X X X X X
X X X X Q X X X
X X X X X X Q X
X X X Q X X X X

X X X X X X X Q
X X X Q X X X X
Q X X X X X X X
X X Q X X X X X
X X X X X Q X X
X Q X X X X X X
X X X X X X Q X
X X X X Q X X X

The total number of ways is : 92
```

NOTE : Only 2 are taken as it is difficult to place all 92

❖ Board Size : 3

```
Enter the size of the board : 3
The total number of ways is : 0
```

❖ Board Size : -1 (Exception handling)

```
Enter the size of the board : -1
ERROR : Invalid board size
```

2) Knight Tour :

❖ Input

```
PS F:\Ganesh\Amrita\Subjects\Sem 2\Data Structures and Algorithms\Project\Group_14_Batch_A> java Knight_Tour.java
Enter the size of the board : █
```

❖ Board Size : 3

```
Enter the size of the board : 3
The total number of knight tours are : 0
```

❖ Board Size : 5

```
The way 300 is :  
[1, 18, 13, 8, 3]  
[12, 5, 2, 19, 14]  
[17, 22, 7, 4, 9]  
[6, 11, 24, 15, 20]  
[23, 16, 21, 10, 25]
```

```
The way 301 is :  
[1, 16, 21, 6, 3]  
[10, 5, 2, 15, 20]  
[17, 22, 11, 4, 7]  
[12, 9, 24, 19, 14]  
[23, 18, 13, 8, 25]
```

```
The way 302 is :  
[1, 16, 11, 6, 3]  
[10, 5, 2, 21, 12]  
[15, 22, 17, 4, 7]  
[18, 9, 24, 13, 20]  
[23, 14, 19, 8, 25]
```

```
The way 303 is :  
[1, 16, 11, 6, 3]  
[10, 5, 2, 17, 12]  
[15, 22, 19, 4, 7]  
[20, 9, 24, 13, 18]  
[23, 14, 21, 8, 25]
```

```
The way 304 is :  
[1, 18, 11, 6, 3]  
[10, 5, 2, 17, 12]  
[19, 22, 13, 4, 7]  
[14, 9, 24, 21, 16]  
[23, 20, 15, 8, 25]
```

```
The total number of knight tours are : 304
```

❖ Board Size : -7

```
Enter the size of the board : -7  
ERROR : Invalid board size
```

3) Maze Challenge :

❖ Input

```
PS F:\Ganesh\Amrita\Subjects\Sem 2\Data Structures and Algorithms\Project\Group_14_Batch_A> java Maze.java
Enter the size (example : '<x><space><y>' where x and y are the row size and column size respectively):
```

❖ Board Size : 3 4

```
Path 34 :
[1, 2, 3, 0]
[6, 5, 4, 0]
[7, 8, 9, 10]
Path : RRDLLDRRR

Path 35 :
[1, 2, 3, 4]
[0, 0, 0, 5]
[0, 0, 0, 6]
Path : RRRDD

Path 36 :
[1, 2, 3, 4]
[0, 0, 6, 5]
[0, 0, 7, 8]
Path : RRRDLDR

Path 37 :
[1, 2, 3, 4]
[0, 7, 6, 5]
[0, 8, 9, 10]
Path : RRRDLLDRR

Path 38 :
[1, 2, 3, 4]
[8, 7, 6, 5]
[9, 10, 11, 12]
Path : RRRDLLLLDRRR

The total number of ways are : 38
```

❖ Board Size : 5 2

```
Path 13 :
[1, 2]
[4, 3]
[5, 0]
[6, 0]
[7, 8]
Path : RDLDDDR

Path 14 :
[1, 2]
[4, 3]
[5, 0]
[6, 7]
[0, 8]
Path : RDLDDRDRD

Path 15 :
[1, 2]
[4, 3]
[5, 6]
[0, 7]
[0, 8]
Path : RDLDRDD

Path 16 :
[1, 2]
[4, 3]
[5, 6]
[8, 7]
[9, 10]
Path : RDLDRDLDR

The total number of ways are : 16
```

❖ Board Size : -1 4

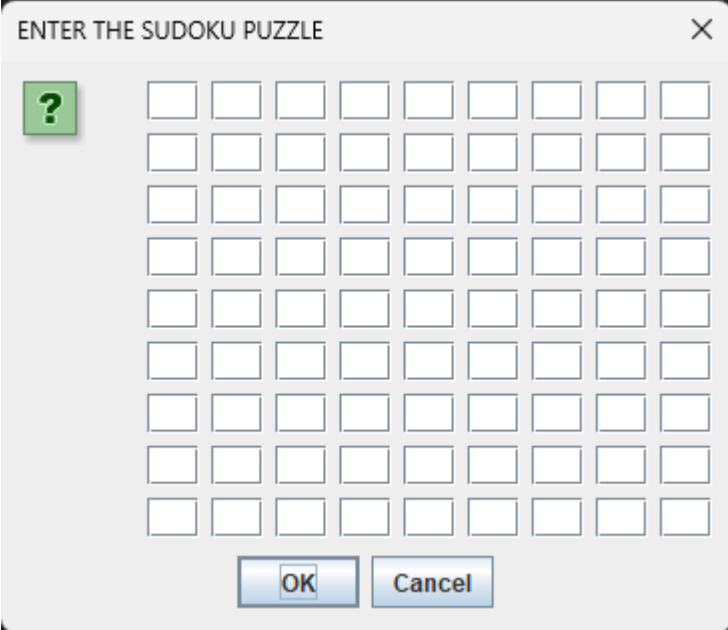
```
Enter the size (example : '<x><space><y>' where x and y are the row size and column size respectively): -1 4
Please enter the correct size. Values lesser than 0 are undefined for this purpose.
```

❖ Board Size : -3 -6

```
Enter the size (example : '<x><space><y>' where x and y are the row size and column size respectively): -3 -6
Please enter the correct size. Values lesser than 0 are undefined for this purpose.
```


4) Sudoku (in java) :

❖ Input



❖ Case 1 : Empty

```
Your puzzle is :
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X
X X X X X X X X X

The solution is :
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[4, 5, 6, 7, 8, 9, 1, 2, 3]
[7, 8, 9, 1, 2, 3, 4, 5, 6]
[2, 1, 4, 3, 6, 5, 8, 9, 7]
[3, 6, 5, 8, 9, 7, 2, 1, 4]
[8, 9, 7, 2, 1, 4, 3, 6, 5]
[5, 3, 1, 6, 4, 2, 9, 7, 8]
[6, 4, 2, 9, 7, 8, 5, 3, 1]
[9, 7, 8, 5, 3, 1, 6, 4, 2]
```

❖ Case 2 : Error Puzzle

```

Your puzzle is :
1 X X X X 1 X X X
X X X X X X X X X
X X 1 X X X X 4 X
X X X X X X X X X
X X X X X X X X X
X X X 5 X X X X X
3 X X X X X X X X
X X X X X X X 9 X
X X X X X X X X X
ERROR : Please re-enter the puzzle correctly
  
```

There is an error because the puzzle has 1 in the same row and 1st 3 x 3 box.

❖ Case 3 : Sample Puzzle

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

```

Your puzzle is :
5 3 X X 7 X X X X
6 X X 1 9 5 X X X
X 9 8 X X X X 6 X
8 X X X 6 X X X 3
4 X X 8 X 3 X X 1
7 X X X 2 X X X 6
X 6 X X X X 2 8 X
X X X 4 1 9 X X 5
X X X X 8 X X 7 9

The solution is :
[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]

```

5) Sudoku (in c) :

❖ Input

```
Enter the Sudoku puzzle (9x9 grid):
```

❖ Case 1 : error puzzle

```

Enter the Sudoku puzzle (9x9 grid):
0 0 0 8 0 9 0 0
0 0 9 0 0 0 0 0
8 9 0 0 0 0 0 7
8 9 9 0 0 0 0 0
5 6 7 9 0 0 7 8
0 8 9 0 7 0 6 0
8 9 0 9 7 8 9 9
7 8 9 0 0 0 6 9
9 0 8 7 9 0 6 7
0 9 8 0 7 6 8 0
9 0 7 8 9 0 8 9

c:\Users\91944\OneDrive\Desktop\c program>cd "c:\Users\91944\OneDrive\Desktop\c program\" && gcc tempCodeRunnerFile.c -o tempCodeRunnerFile && "c:\Users\91944\OneDrive\Desktop\c program\tempCodeRunnerFile

```

❖ Case 2 : sample puzzle

Enter the Sudoku puzzle (9x9 grid):

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

The solution is :

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

ANALYSIS AND DISCUSSION :

- **Time complexity analysis :**

1. Nqueens : n^n , n = size of board
2. Knight tour : n^n , n = size of board
3. Maze challenge : $4^{xy-1} = 4^{xy}$, $(x,y) = (\text{rows}, \text{columns})$
4. Sudoku solver : 9^n , n = number of empty cells

- **Limitations of the program :**

Mostly, backtracking algorithms has a huge time complexity. When the sample space is high and the worst case is considered, it takes a huge time which should be avoided. It even has a high space complexity. So, the program takes a lot of space. This shows us that this may not be the most efficient algorithm. This can solve only certain types of problems which has a definite search interval. For search intervals till infinity, it does not work properly. It does not throw an error, but it runs till it finds the solution.

- **Overcoming the limitations :**

While we can't completely eliminate the limitations shown above, we can reduce it to a extent so that the program becomes much usable. We can make the algorithms program specific so that we don't use space and time unnecessarily. We can optimize the same program to use less space by avoiding certain lines of code when not necessary. Also, we can use multithreading provided by java to run multiple threads at the same time and solve the problems much faster. We can also combine backtracking algorithms with other algorithms so that the program becomes optimized.

APPENDIX :

1) Nqueens :

```
import java.util.*;
import java.lang.Math;

public class Nqueens extends Thread{

    static int count = 0;
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the board : ");
        int n = sc.nextInt();

        if(n<=0){
            System.out.println("ERROR : Invalid board size");
            System.exit(1);
        }

        boolean[][] board = new boolean[n][n];
        get(board,0);

        System.out.println("The total number of ways is : "+count);
        System.out.print("Program will exit in a moment...");
        try{
            Thread.sleep(50000);
        }
        catch(Exception e){}
    }

    public static void get(boolean[][] board,int row){

        if(row == board.length){
            display(board);
            System.out.println();
            count++;
            return;
        }

        for(int i=0;i<board.length;i++){
            if(check(board,row,i)){
                board[row][i] = true;
                get(board,row+1);
                board[row][i] = false;
            }
        }
    }
}
```

```

    }

    private static boolean check(boolean[][] board, int row, int col) {

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

        int lmin = Math.min(row,col),rmin = Math.min(row,board.length-col-1);

        int r = row,c = col;

        for(int j=0;j<lmin;j++){
            row--;
            col--;
            if(board[row][col]){
                return false;
            }
        }

        for(int j=0;j<rmin;j++){
            r--;
            c++;
            if(board[r][c]){
                return false;
            }
        }

        return true;
    }

    public static void display(boolean[][] board) {
        for(boolean[] tem : board){
            for(boolean temp : tem){
                if(temp == true){
                    System.out.print("Q ");
                }
                else{
                    System.out.print("X ");
                }
            }
            System.out.println();
        }
    }
}

```

2) Knight Tours :

```
import java.util.*;

public class Knight_Tour extends Thread{
    public static int count=0;
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the board : ");
        int n = sc.nextInt();

        if(n<=0){
            System.out.println("ERROR : Invalid board size");
            System.exit(1);
        }

        int[][] board = new int[n][n];

        solve(board,0,0,1);

        System.out.println("The total number of knight tours are : "+count);
        System.out.print("Program will exit in a moment...");
        try{
            Thread.sleep(50000);
        }
        catch(Exception e){}

    }

    public static void solve(int[][] board, int a, int b,int step) {

        int n=board.length;
        int[][] values = {{2,2,1,1,-2,-2,-1,-1},{1,-1,2,-2,1,-1,2,-2}};

        board[a][b] = step;

        if(step>=n*n){
            board[a][b] = step;
            System.out.println("The way "+(count+1)+" is : ");
            display(board);
            System.out.println();
            count++;
        }

        for(int i=0;i<8;i++){
            int a1 = a+values[0][i];
            int b1 = b+values[1][i];
```



```
        if(a1<board.length && a1>=0 && b1<board.length && b1>=0 &&
board[a1][b1] == 0){
            solve(board,a1,b1,step+1);
        }
    }

    board[a][b] = 0;

}

public static void display(int[][] board){

    for(int[] row : board){
        System.out.println(Arrays.toString(row));
    }

}

}
```

3) Maze Challenge :

```
import java.util.*;

public class Maze extends Thread{

    static int count=0;

    static void calculate(int a,int b,boolean[][] array,String output,int
x,int y,int[][] path,int step){

        int c=x,d=y;
        boolean[][] temp = array;

        if(c==(a-1) && d==(b-1)){
            System.out.println("Path "+(count+1)+" :");
            for(int[] arr : path ){
                System.out.println(Arrays.toString(arr));
            }
            System.out.println("Path : "+output+"\n");
            count++;
            return;
        }

        if(c<a-1){
            c++;
            if(array[c][d] != false){
                array[c][d] = false;
                path[c][d] = step;
                calculate(a,b,array,output+"D",c,d,path,step+1);
                path[c][d] = 0;
                array[c][d] = true;
            }
        }

        c=x;
        d=y;

        if(d<b-1){
            d++;
            if(array[c][d] != false){
                array[c][d] = false;
                path[c][d] = step;
                calculate(a,b,array,output+"R",c,d,path,step+1);
                path[c][d] = 0;
                array[c][d] = true;
            }
        }
    }
}
```

```

        c=x;
        d=y;

        if(d>0){
            d--;
            if(array[c][d] == true){
                array[c][d] = false;
                path[c][d] = step;
                calculate(a,b,array,output+"L",c,d,path,step+1);
                path[c][d] = 0;
                array[c][d] = true;
            }
        }

        c=x;
        d=y;

        if(c>0){
            c--;
            if(array[c][d] == true){
                array[c][d] = false;
                path[c][d] = step;
                calculate(a,b,array,output+"U",c,d,path,step+1);
                path[c][d] = 0;
                array[c][d] = true;
            }
        }
    }

    public static void main(String args[]){

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size (example : '<x><space><y>' where
x and y are the row size and column size respectively): ");
        int a,b;

        a=sc.nextInt();
        b=sc.nextInt();

        if(a<=0 || b<=0){
            System.out.println("Please enter the correct size. Values
lesser than 0 are undefined for this purpose.");
            System.exit(1);
        }

        boolean[][] array = new boolean[a][b];

```

```
        for(int i=0;i<a;i++){
            for(int j=0;j<b;j++){
                array[i][j] = true;
            }
        }

        array[0][0] = false;
        int[][] path= new int[a][b];
        path[0][0] = 1;

        System.out.println("\nThe paths are : \n");
        calculate(a,b,array,"",0,0,path,2);

        System.out.println("\nThe total number of ways are : "+count);
        System.out.print("Program will exit in a moment...");
        try{
            Thread.sleep(50000);
        }
        catch(Exception e){}
    }
}
```

4) Sudoku solver (java) :

```
import java.util.ArrayList;
import java.util.List;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import java.awt.GridLayout;

import java.util.*;

public class Sudoku extends Thread{

    static int i=0,j=0;
    static int[][] array = new int[9][9];
    public static final int OPTIONS = 81;
    public List<JTextField> textfields = new ArrayList<>(OPTIONS);

    private JPanel panel;

    public static void main(String[] args) {
        createAndShowGUI();

        int k,l;
        System.out.println("Your puzzle is : ");
        for(k=0;k<9;k++){
            for(l=0;l<9;l++){
                if(array[k][l] != 0){
                    System.out.print(array[k][l] + " ");
                }
                else{
                    System.out.print("X ");
                }
            }
            System.out.println();
        }

        for(int i=0 ; i<9 ; i++){
            for(int j=0;j<9;j++){
                int value = array[i][j];
                array[i][j] = 0;
                if(value==0){
                    continue;
                }
                if(!(check(array,i,j,value))){
                    System.out.println("ERROR : Please re-enter the puzzle
correctly");
                    try{
```

```

        Thread.sleep(5000);
    }
    catch(Exception e){}
    System.exit(1);
}
array[i][j] = value;
}
}

solve(array,0,0);

System.out.println("ERROR : Cannot solve the puzzle, please check the
values given.");
}

public Sudoku(){
    panel = new JPanel();
    panel.setLayout(new GridLayout(9,9,-10,6));
    for(int i =0;i< OPTIONS;i++){
        JTextField textfield = new JTextField(2);
        textfields.add(textfield);
        panel.add(new JLabel());
        panel.add(textfield);
    }
}

public static void createAndShowGUI() {
    Sudoku example = new Sudoku();
    int result = JOptionPane.showConfirmDialog(null, example.panel,
        "ENTER THE SUDOKU PUZZLE", JOptionPane.OK_CANCEL_OPTION);
    if (result == JOptionPane.OK_OPTION) {
        i=0;
        j=0;
        for(JTextField textfield : example.textfields){

            if(textfield.getText().length() == 0){
                array[i][j] = 0;
            }
            else{
                int a1,a2,a3,a4,a5,a6,a7,a8,a9;
                String temp = textfield.getText();

                a1 = temp.compareTo("1");
                a2 = temp.compareTo("2");
                a3 = temp.compareTo("3");
                a4 = temp.compareTo("4");
                a5 = temp.compareTo("5");
            }
        }
    }
}

```

```

        a6 = temp.compareTo("6");
        a7 = temp.compareTo("7");
        a8 = temp.compareTo("8");
        a9 = temp.compareTo("9");

        if(a1 != 0 && a2 != 0 && a3 != 0 && a4 != 0 && a5 != 0 &&
a6 != 0 && a7 != 0 && a8 != 0 && a9 != 0 ){
            System.out.println("ERROR : Enter the correct integer
values between 1 and 9");
            System.exit(0);
        }
        else{
            array[i][j] = Integer.parseInt(temp);
        }

        if(j<8){
            j++;
        }
        else{
            i++;
            j=0;
        }
    }

}

}

public static void solve(int[][] board, int a, int b) {

    int i,j,k=0;

    for(i=0;i<9;i++){
        for(j=0;j<9;j++){
            if(board[i][j] != 0){
                k++;
            }
        }
    }

    if(k==81){
        System.out.println("\nThe solution is : ");
        for(int[] row : board){
            System.out.println(Arrays.toString(row));
        }
        System.out.println("This program will exit in a moment...");
        try{
            Thread.sleep(50000);

```

```

    }
    catch(Exception e){}
    System.exit(0);
}

if(board[a][b] != 0){
    if(b<8){
        solve(board,a,b+1);
        return;
    }
    else{
        solve(board,a+1,0);
        return;
    }
}

if(board[a][b] == 0){
    for(i=1;i<=9;i++){
        if(check(board,a,b,i)){
            board[a][b] = i;
            if(b<8){
                solve(board,a,b+1);
            }
            else{
                solve(board,a+1,0);
            }
            board[a][b] = 0;
        }
    }
}

}

public static boolean check(int[][] board, int a, int b, int val) {

    int i,j;
    for(i=0;i<9;i++){
        if(board[a][i] == val || board[i][b] == val){
            return false;
        }
    }

    int c= (a/3) + 1;
    int d = (b/3) + 1;

    for(i=(c-1)*3 ; i< c*3 ;i++){
        for(j=(d-1)*3 ; j<d*3 ;j++){
            if(board[i][j] == val){

```



```
        return false;
    }
}

return true;
}
```

5) Sudoku solver (c) :

```
#include <stdio.h>
#include <stdlib.h>

void solve(int board[9][9], int a, int b);
int check(int board[9][9], int a, int b, int val);

int main() {
    int board[9][9];

    printf("Enter the Sudoku puzzle (9x9 grid):\n");
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            int num;
            if (scanf("%d", &num) != 1) {
                printf("Invalid input. Please enter numbers only.\n");
                return 1;
            }

            if (num < 0 || num > 9) {
                printf("Invalid input. Please enter numbers from 0 to 9\n");
                return 1;
            }

            board[i][j] = num;
        }
    }

    solve(board, 0, 0);

    return 0;
}

void solve(int board[9][9], int a, int b) {
    int i, j, k = 0;

    for (i = 0; i < 9; i++) {
        for (j = 0; j < 9; j++) {
            if (board[i][j] != 0) {
                k++;
            }
        }
    }

    if (k == 81) {
```

```

        printf("\nThe solution is :\n");
        for (i = 0; i < 9; i++) {
            for (j = 0; j < 9; j++) {
                printf("%d ", board[i][j]);
            }
            printf("\n");
        }
        exit(0);
    }

    if (board[a][b] != 0) {
        if (b < 8) {
            solve(board, a, b + 1);
            return;
        } else {
            solve(board, a + 1, 0);
            return;
        }
    }

    if (board[a][b] == 0) {
        for (i = 1; i <= 9; i++) {
            if (check(board, a, b, i)) {
                board[a][b] = i;
                if (b < 8) {
                    solve(board, a, b + 1);
                } else {
                    solve(board, a + 1, 0);
                }
                board[a][b] = 0;
            }
        }
    }
}

int check(int board[9][9], int a, int b, int val) {
    int i, j;
    for (i = 0; i < 9; i++) {
        if (board[a][i] == val || board[i][b] == val) {
            return 0;
        }
    }

    int c = (a / 3) + 1;
    int d = (b / 3) + 1;

    for (i = (c - 1) * 3; i < c * 3; i++) {
        for (j = (d - 1) * 3; j < d * 3; j++) {

```

```
        if (board[i][j] == val) {  
            return 0;  
        }  
    }  
}  
return 1;  
}
```