Recursion and backtracking

1. You are given a number n, representing the number of rows.
2. You are given a number m, representing the number of columns.
3. You are given n*m numbers, representing elements of 2d array a. The numbers can be 1 or 0 only.
4. You are standing in the top-left corner and have to reach the bottom-right corner. Only four moves are allowed 't' (1-step up), 'l' (1-step left), 'd' (1-step down) 'r' (1-step right). You can only move to cells which have 0 value in them. You can't move out of the boundaries or in the cells which have value 1 in them (1 means obstacle)
5. Print all paths that can be used to move from top-left to bottom-right.

Sample Input 0

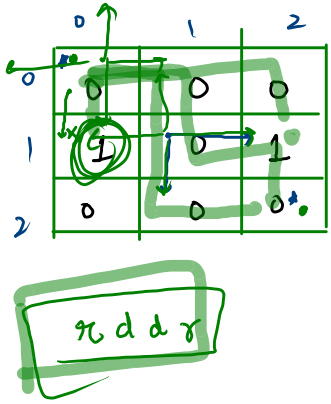
```
3 3
0 0 0
1 0 1
0 0 0
```

Sample Output 0

```
rddr
```

t r l d

r l d t



n = 3

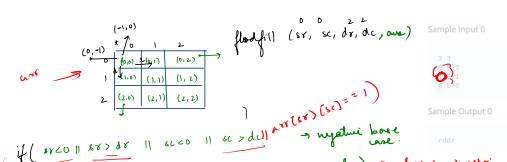
m = 3

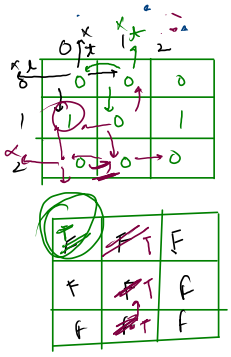
t ↑

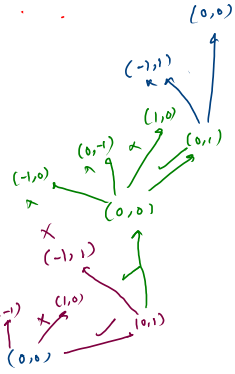
l ←

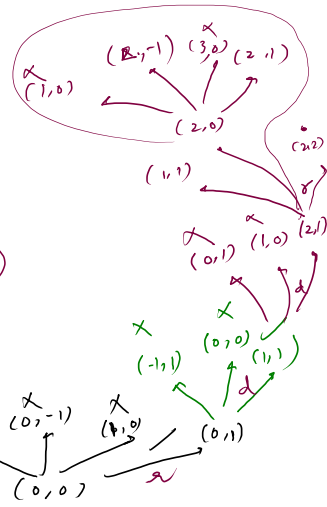
d ↓

r →









⇒ We go into the infinite recursion

↳ soln:- create a boolean, filled with false initially.

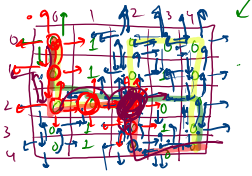
↳ false → not go to that call.

↳ True → you already go to that call, choose another one.

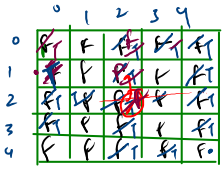
10 min

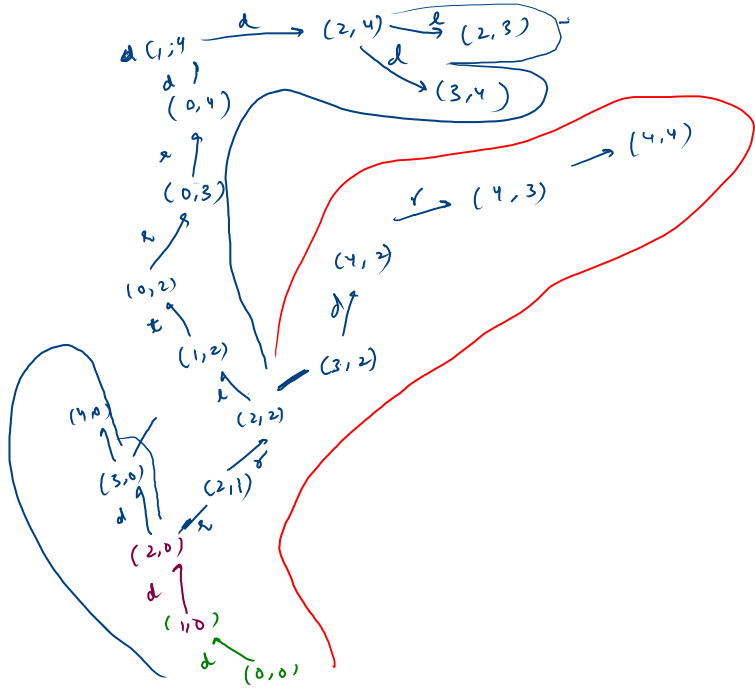
20. 58

```
ff( checkpath, sr, sc, dr, dc, ans ){
  if( sr<0 || sc<0 || sr>dr || sc>dc || arr[sr][sc]==1 || checkpath[sr][sc]==true )
      return;
  if( sr == dr && sc == dc)
      bypass(ans);
      return;
  checkpath[sr][sc] = true;
  ff( checkpath, sr-1, sc, dr, dc, ans+"u");
  ff( checkpath, sr, sc-1, dr, dc, ans+"l");
  ff( checkpath, sr+1, sc, dr, dc, ans+"d");
  ff( checkpath, sr, sc+1, dr, dc, ans+"r");
}
```



4 3 ans





```java
import java.io.*;
import java.util.*;

public class Solution {
    public static void floodFill(boolean checkCell[][], int arr[][], int sr, int sc, int dr, int dc, String ans){
        if(sr<0 || sc<0 || sr>dr || sc>dc || arr[sr][sc] == 1 || checkCell[sr][sc] == true){
            return;
        }

        if(sr == dr && sc == dc){
            System.out.println(ans);
            return;
        }

        checkCell[sr][sc] = true;

        floodFill(checkCell, arr,sr-1,sc,dr,dc,ans+"t");
        floodFill(checkCell, arr,sr,sc-1,dr,dc,ans+"l");
        floodFill(checkCell, arr,sr+1,sc,dr,dc,ans+"d");
        floodFill(checkCell, arr,sr,sc+1,dr,dc,ans+"r");

        checkCell[sr][sc] = false;

    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. */
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int m = scn.nextInt();
        int arr [][] = new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                arr[i][j] = scn.nextInt();
            }
        }

        boolean checkCell [][] = new boolean [n][m];

        floodFill(checkCell,arr, 0, 0, n-1,m-1,"");
    }
}
```

→ Avoid infinite recursion

→ Avoid blocking of other paths

N Queens (Day 27)

Problem | Submissions | Leaderboard | Discussions

1. You are given a number n, the size of a chess board.
2. You are required to place n number of queens in the n * n cells of board such that no queen can kill another. Note - Queens kill at distance in all 8 directions
3. Calculate and print all safe configurations of n-queens. Use sample input and output to get more idea.

Sample Input 0

```
4
```

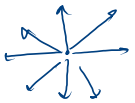
Sample Output 0

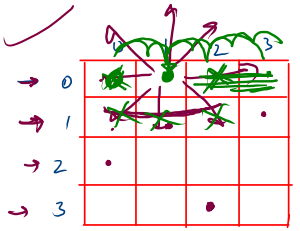
```
0-1, 1-3, 2-0, 3-2, .
0-2, 1-0, 2-3, 3-1, .
```

Can it is possible that I can fill row 1 before row 0?

row = 0 1 2 3

8 → 3 position

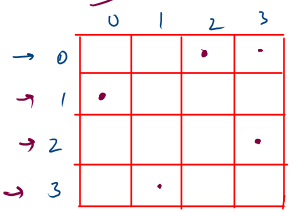




nqueen(chessboard, row, n, safe) {

if(row == n)

Syso(ans)



row = 0 1 2 3

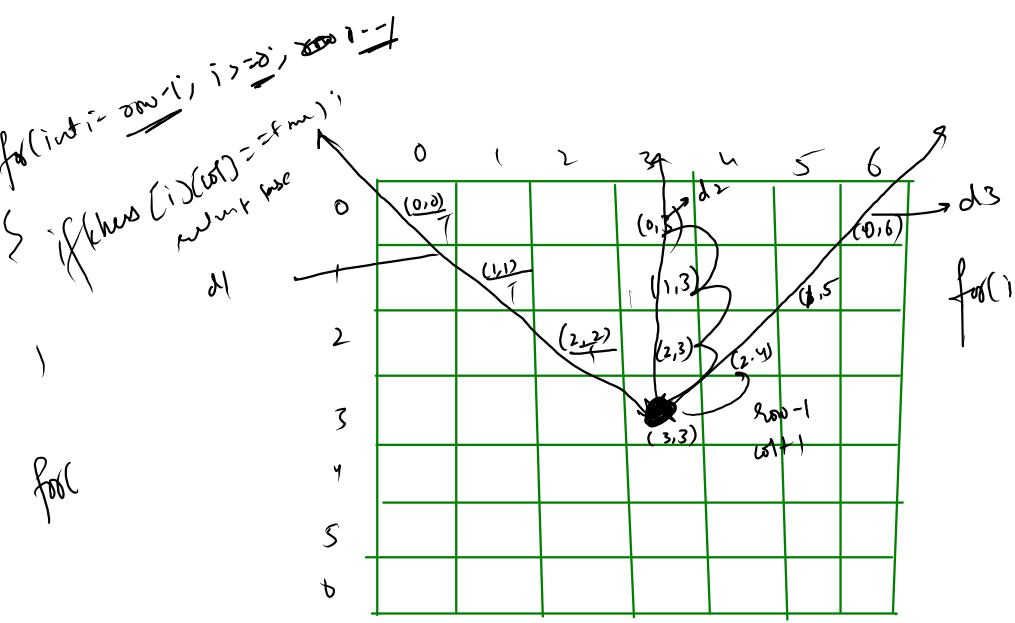
col =

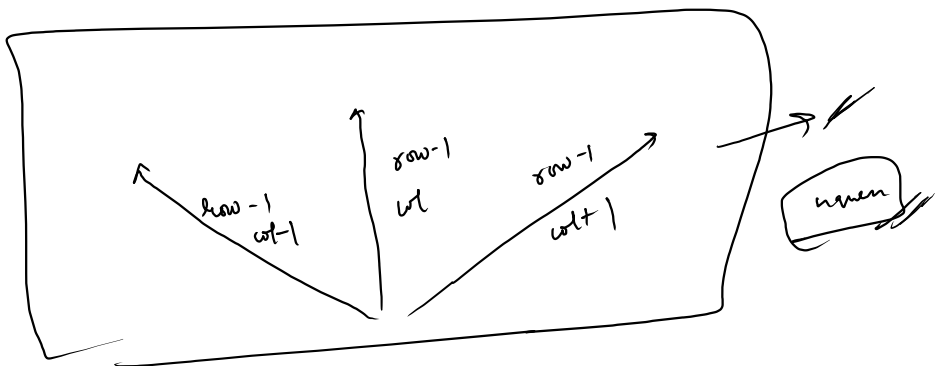
row = 0 1 2 3

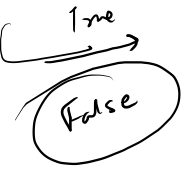
col → n

→ chessboard[row][col] = true

chessboard[row][col] = false


0 1 2 3 4 5 6
0
2
3
4
5
6
(0,0)
(1,1)
(2,2)
(0,3)
(1,3)
(2,3)
(2,4)
(0,5)
(0,6)
(3,3)
d1
d2
d3
row-1
col+1


row-1
col-1
row-1
col
row-1
col+1




```java
public class Solution {
    public static boolean isSafeQueen(boolean chessBoard[][], int row, int col){

        // For d1
        for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--){
            if(chessBoard[i][j] == true) return false;
        }

        // For d2
        for(int i=row-1;i>=0;i--){
            if(chessBoard[i][col] == true) return false;
        }

        // For d3
        for(int i=row-1,j=col+1;i>=0 && j<chessBoard.length;i--,j++){
            if(chessBoard[i][j] == true) return false;
        }

        return true; // It is the safe position to place queen
    }
    public static void nqueen(boolean chessBoard[][], int row, String ans){

        if(row == chessBoard.length){
            System.out.println(ans + ".");
            return;
        }

        for(int col = 0;col<chessBoard.length;col++){
            if(isSafeQueen(chessBoard,row,col) == true){
                chessBoard[row][col] = true;
                nqueen(chessBoard, row+1, ans + row +"-"+col+", ");
                chessBoard[row][col] = false;
            }
        }

    }
```