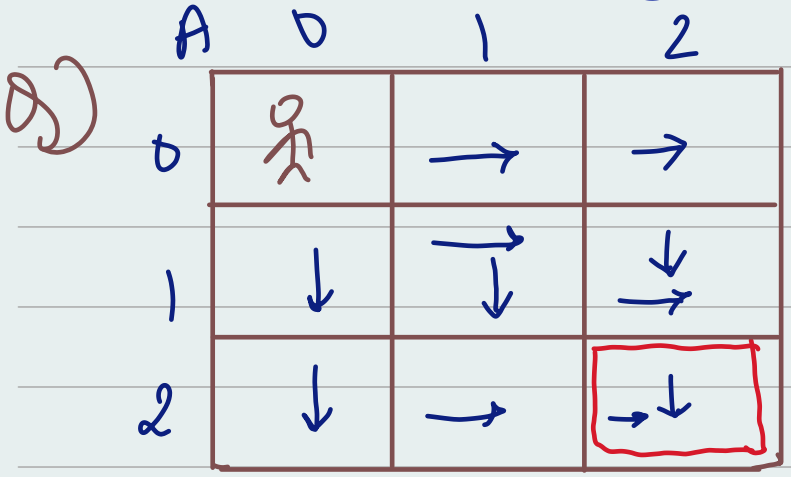
Backtracking & Maze Problems :



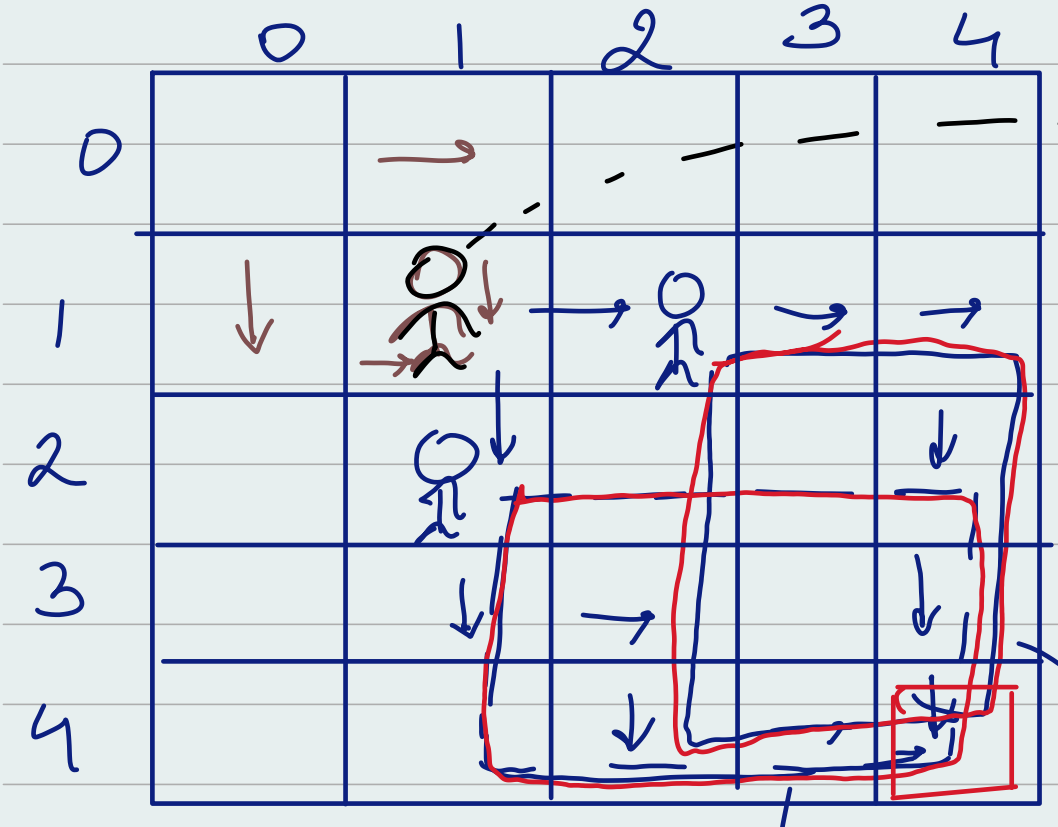
Counting Paths

Right

down

$(0,0) \rightarrow (2,2)$

Ans : R, R, D, D
D, D, R, R
R, D, R, D
D, R, D, R



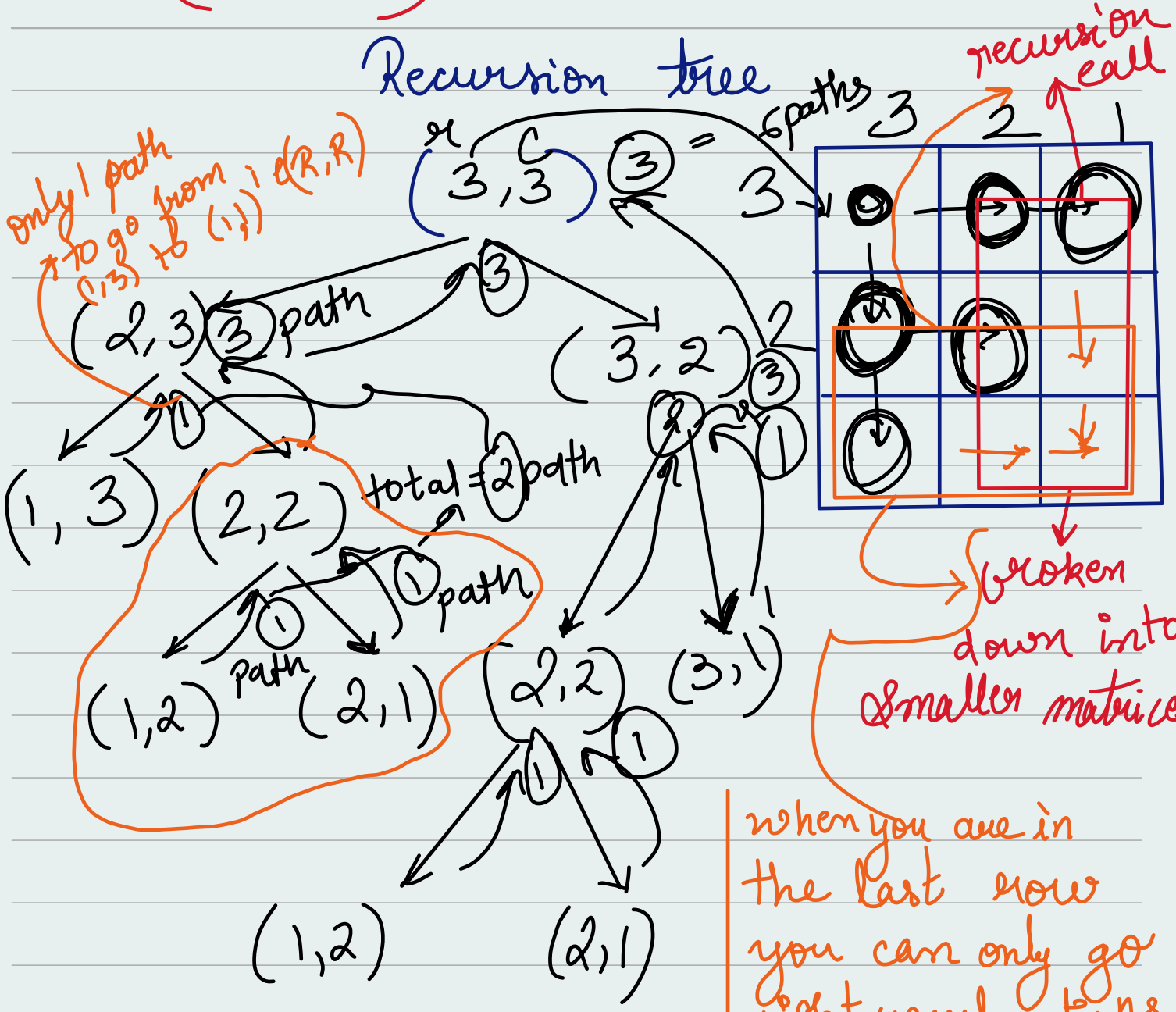
R, D
D, R

Path till that point possible outcomes

Recursion call

(P&P) Recursion call:

Recursion tree



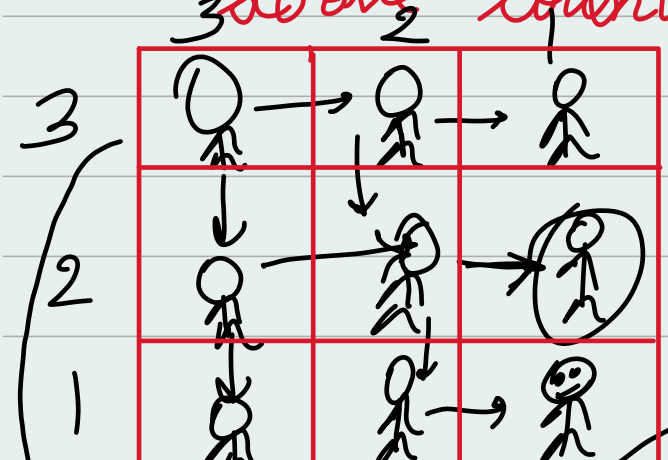
// base condition whenever the row or column hits 1 return 1

when you are in the last row you can only go rightward dirns. & when you are in the last column you can only go in the downward directions.

Code: psvm () {
System out println (countPaths (3, 3));

```
}

static int countPaths(int row,
                      int column){
    // base condition

    if(row == 1 || column == 1){
        return 1;
    }

    int left = countPath(row-1,
                         column);
    int right = countPath(row,
                          column-1);

    return left + right;
}
}
```

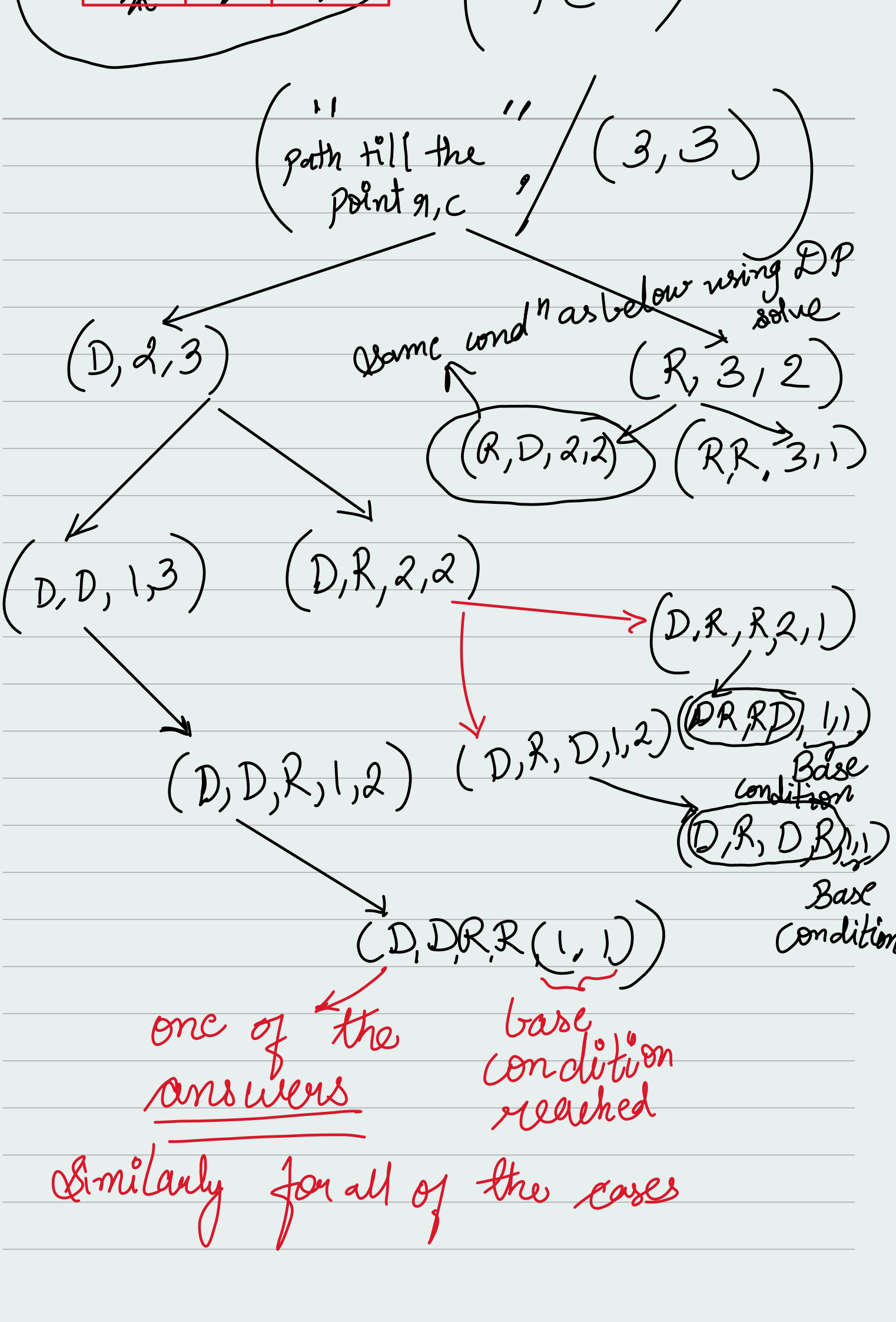
* Now printing the actual path for above counting paths



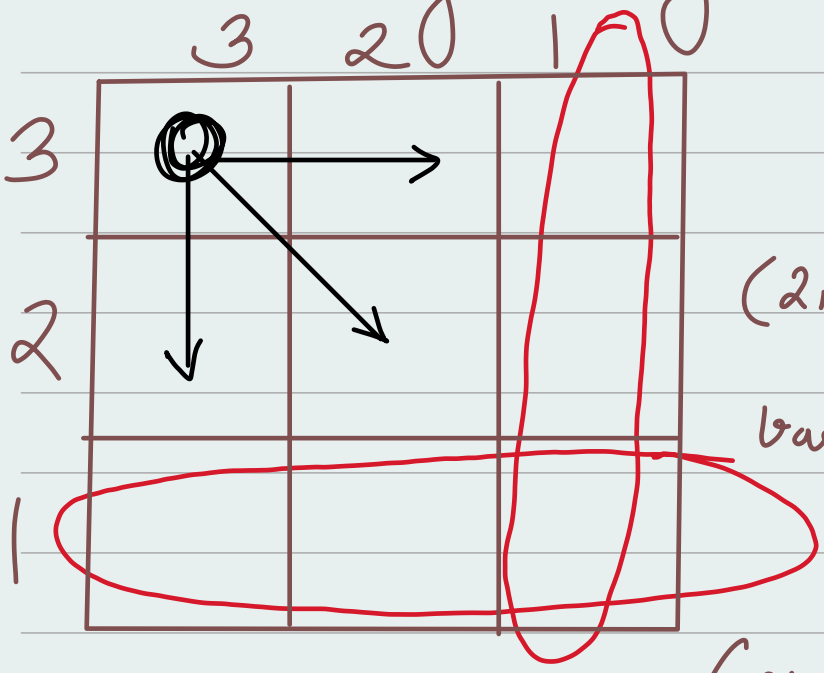
Processed / unprocessed

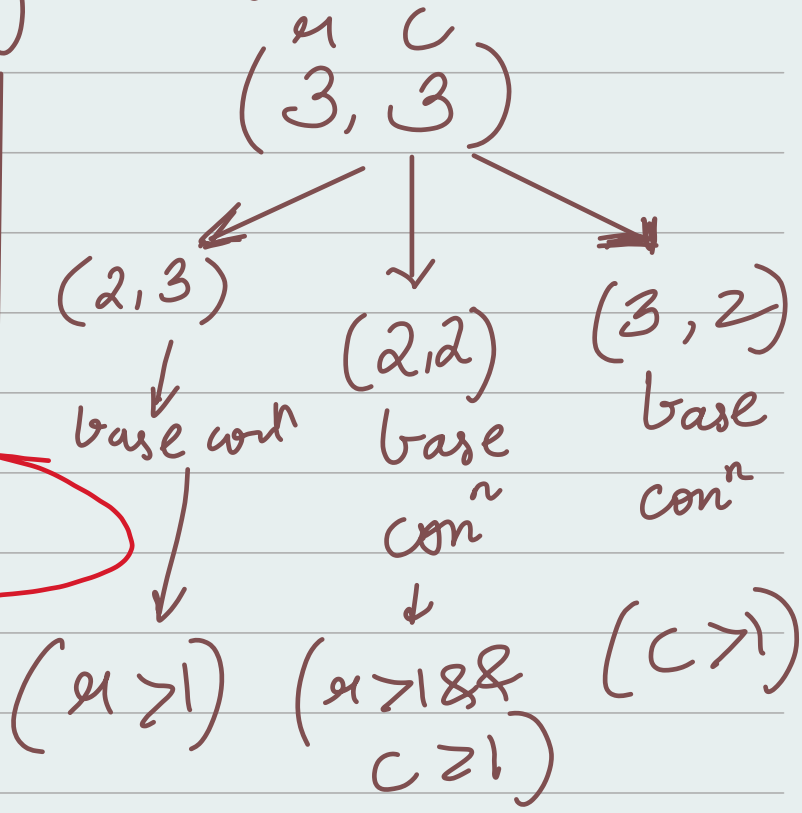
" "

path walked till that point (row, column)

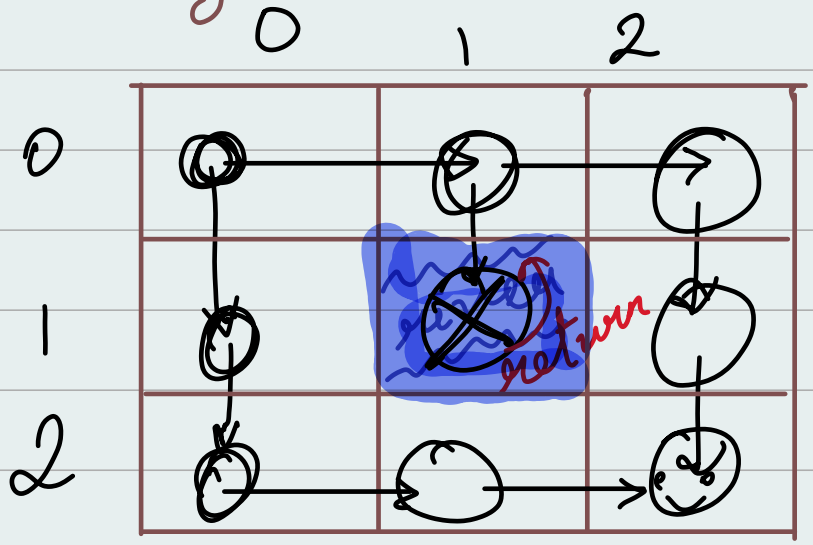

("path till the point r,c", (3,3))
(D,2,3)
same cond^n as below using DP solve
(R,3,2)
(R,D,2,2)
(R,R,3,1)
(D,D,1,3)
(D,R,2,2)
(D,R,R,2,1)
(DR,RD,1,1)
Base condition
(D,D,R,1,2)
(D,R,D,1,2)
(D,R,DR,1,1)
Base condition
(D,DRR,1,1)
one of the answers
base condition reached
Similarly for all of the cases

* Including Diagonal Paths

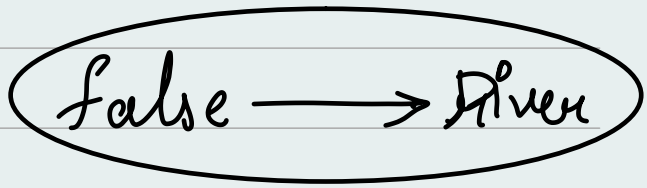


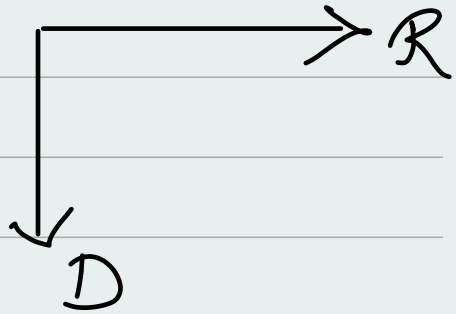


8) Maze with obstacles.



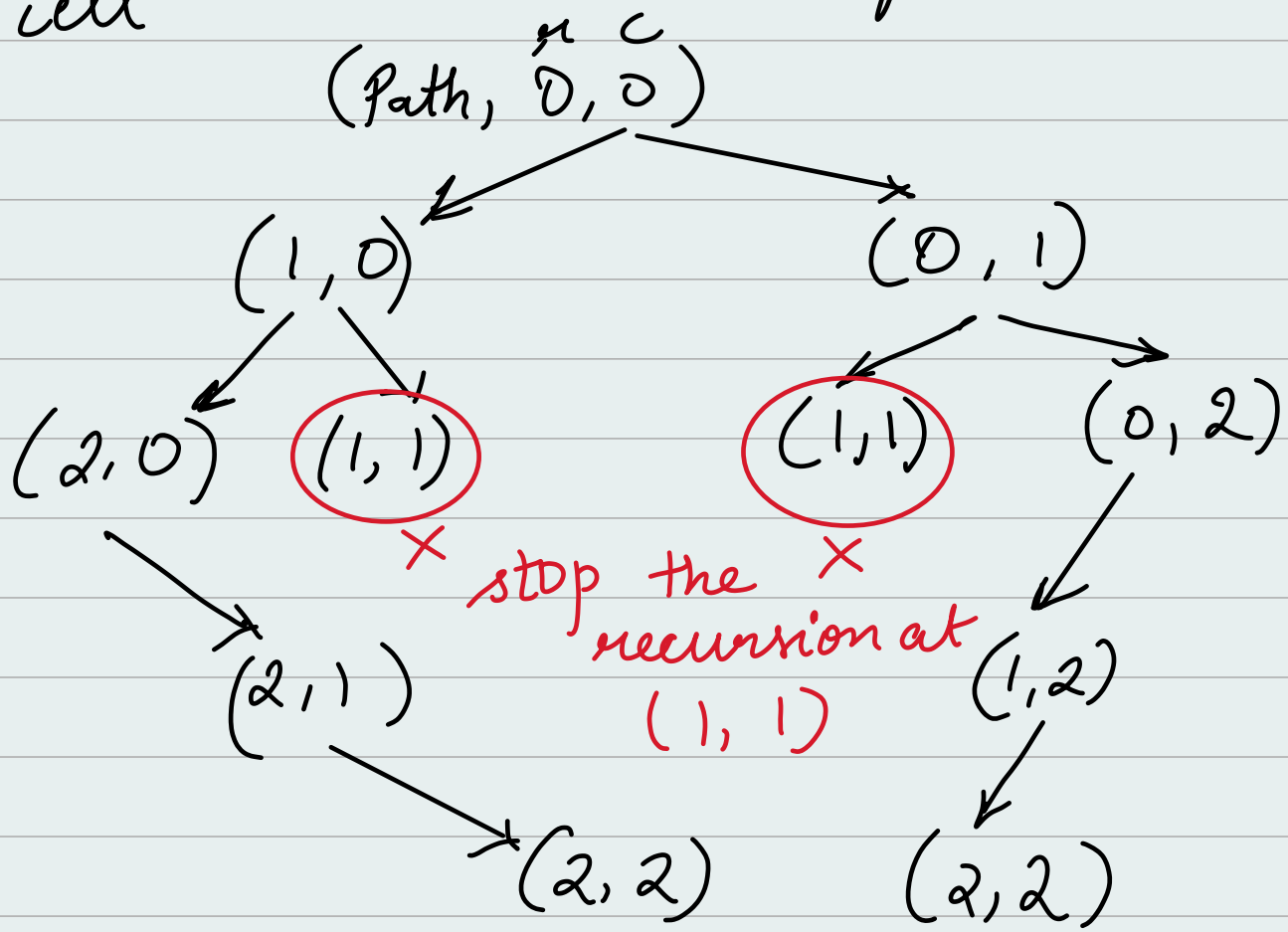
This is boolean matrix





Note! When you land on a new cell, check whether that is river or not.

9) you land on river stop recursion on that call Just return from the river cell



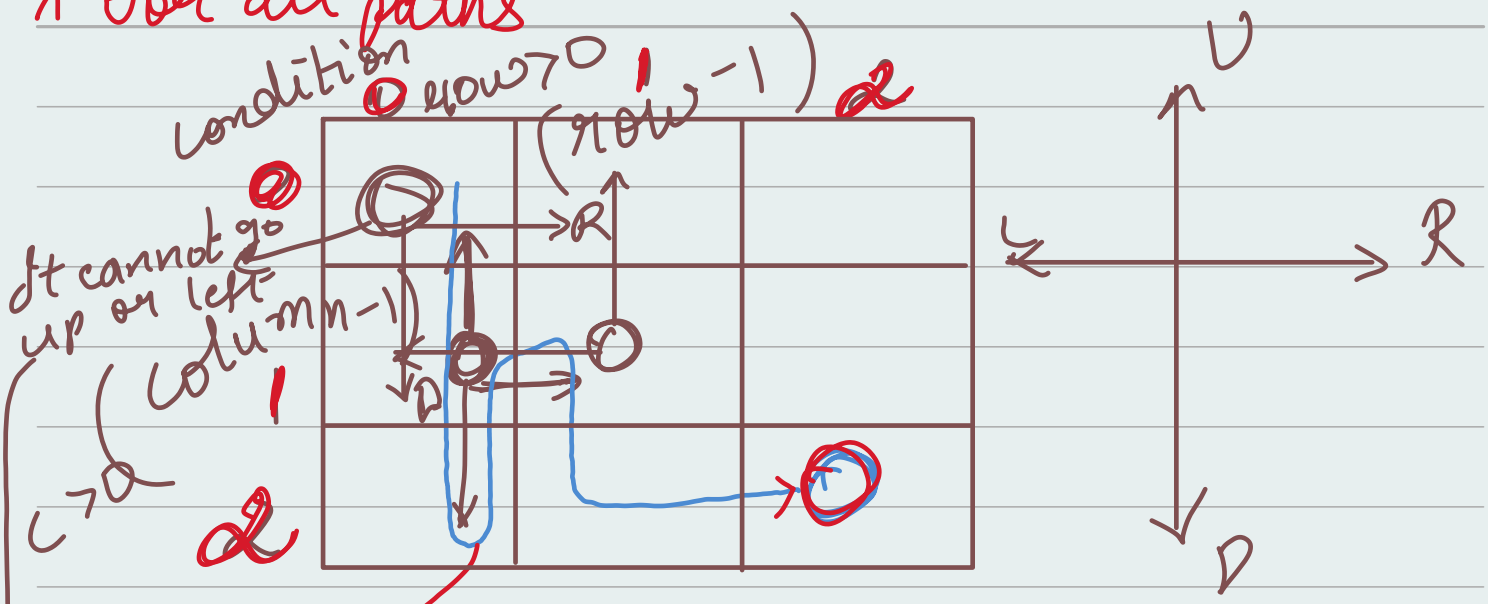
Code:

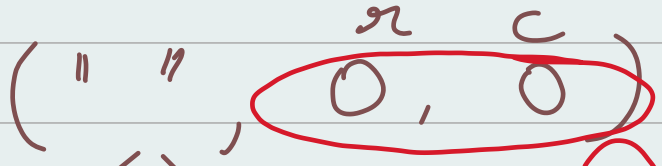
```
ps vm ( String [ ], args ) {
    boolean [ ][ ] board = {
        { true, true, true },
        { true, false, true },
        { true, true, true },
    };
    pathRestrictions ( " ", board, 0, 0);
}
```

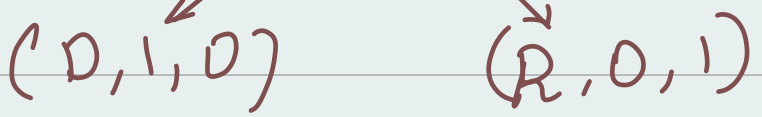
```
static void pathRestrictions(String p,
boolean[][] maze, int row, int col){
    // base condition
    if(row == maze.length-1 && col ==
        maze[0].length-1){
        System.out.println(p);
        return;
    }
    if(!maze[row][col]){ //not false is
                true, then return.
        return;
    }
    if(row < maze.length-1){
    pathRestrictions(p + 'D', maze,
                row+1, column);
    }
    if(column < maze[0].length-1){
    pathRestrictions(p + 'R', maze, row
                column+1);
    }
}
```

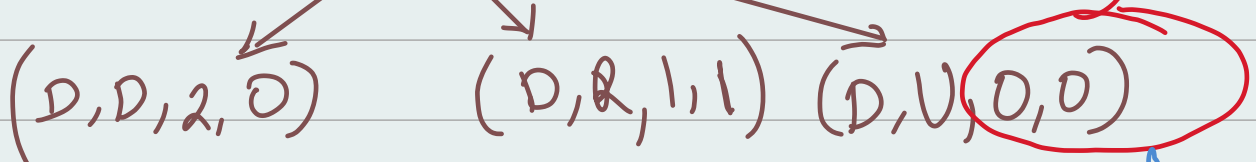
Output: DDRR
RRDD

* For all paths









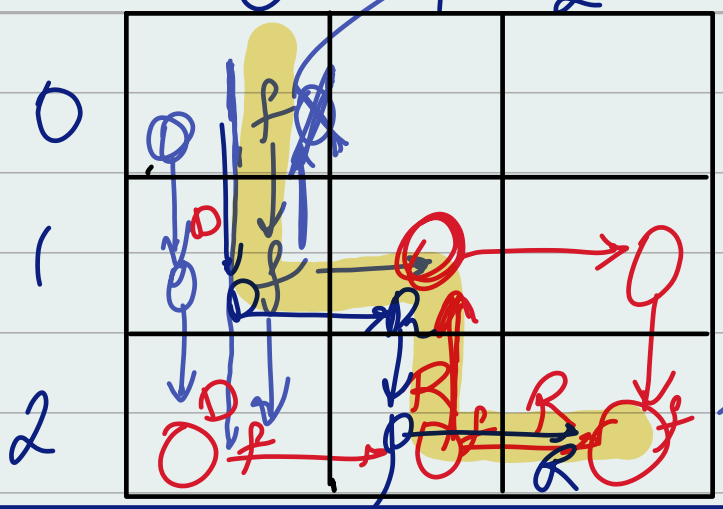
Recursion never ending that is the problem

→ cannot be an answer - Cause the the path cannot be moved back from where it is started It will result in endless loop ie: recursion & Do not move back to the same path. }

* How to solve?

All cells that are visited mark there as false. So that it does not go there Same concept

as the river problem



Already visited all the blocks that are visited like (0,0) (1,0) will return to main function and wont execute to fall into loop.

It should allow to go into the yellow marked path but because it is already marked as false by previous path.

* What is backtracking?

("", 0, 0)

(D, 1, 0) (R, 0, 1)

return up

(D,D,2,0) (D,R,1,1) (D,U,0,0)

up return

(D,D,R,2,1) (D,D,U,1,0)

Right up left

(D,D,R,R,2,2) (D,D,R,U,1,1) (D,D,R,L,1,0)

false

(D,D,R,U,D,2,1) (D,D,R,U,R,1,2)

Ans

(D,D,R,U,R,D,2,2)

(D,R, 1,1)

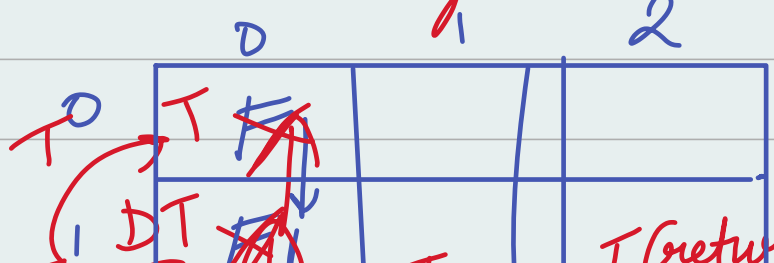
(D,R,D,2,1)

No you cannot

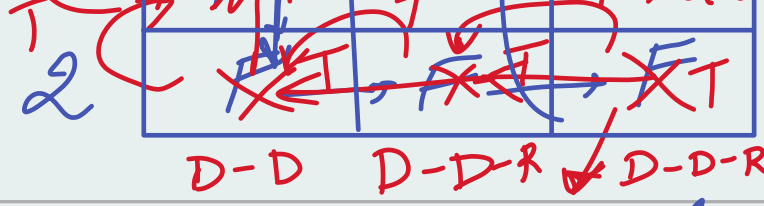
why?

* Common Sense :- Marking false ==> I have that cell in my current path. So when that path is over, ex: you are in another recursion call, those cell should not be false

* While you are moving back, you restore the maze as it was



very basic recursion question



When do we go back?

ans found

It happens when function is returned.

When you have found a potential answer.

When we come out of the recursive function → Now we are in the above recursion call

Hence, Remark the cell as true.

This is known as backtracking.

Change the changes that are made via past recursive calls from false to true.

* While going in the below recursion calls we are making some changes, so when we go outside those recursive calls, the changes that was made via recursion function should also not be available This is known as backtracking.

*Q) Print Matrix and Paths :-

1		
2	5	6

D-D-R-R

D-D-R-U-R-D

While going back through (5) to (4) block, erase the path 5 to 0 Similar concept as above like making true for a false condition

→ As we are going every steps down or every level down for recursion calls the values are getting added like DDRR for 4 level down and DDRURD for 6 level down while going back erase the path number to zero similarly as backtracking.

★ Take a step variable → int step
both into argument fun

★ update the array path → int[][]path

★ print it in base condition.

★ Backtrack

Code:

```
psvm ( ) {
  boolean[][] board = {
    { true, true, true },
```

```
        {true, true, true},
        {true, true, true},
    };
    int[][] path = new int[board.length]
                   [board[0].length];
    allPathPrint("", board, 0, 0,

static void allPathPrint(String p, boolean
    [][] maze, int row, int column,
    int[][] path, int step){
    // base condition
    if(row == maze.length-1 && column ==
        maze.length-1){
        path[row][column] = step;
        for(int[] arr : path)
            System.out.println(Arrays.
            toString(arr));
        }
        System.out.println(p);
        System.out.println();
        return;
    }

    if(!maze[row][column]){//not false
        return,          is true, then return
    }
    maze[row][column] = false;
    path[row][column] = step;
```

```
if(row < maze length - 1){
    allpathprint( p+'D', maze, row+1,
    column , path, &step+1) ;
}
if(column < maze[0]. length - 1){
allPathPrint (p+'R', maze, row, column+1,
        path, &step+1 );
}
if (row > 0 ){
allPathPrint (p+'U', maze, row-1, column,
        path, &step+1);
}
if (column > 0){
    allPathPrint (p+'L', maze, row,
    column-1, path, &step+1);
}
// fun^n over & fun^n gets removed also the
changes made by that particular fun^n gets
removed.

    maze[row][column] = true,
    path[row][column] = 0,
}
}
```