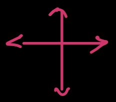
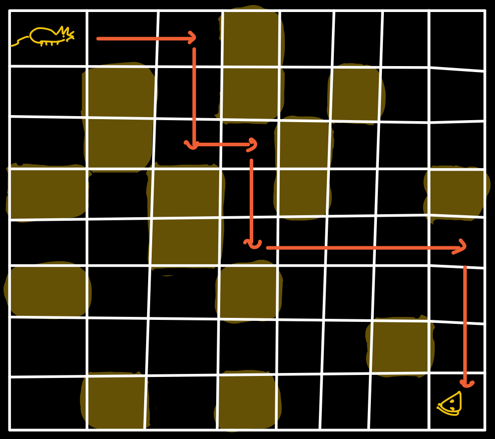
Rat in a maze

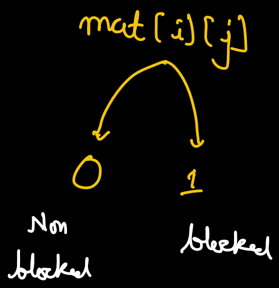




Given the maze.

And initial location of mouse (x,y).

Return true if there exist a path from the rat to the cheese.



```
boolean   mazeSolver ( mat, N, M, x, y) {
          if(x == N-1 && y == M-1) ret true;
          if(x<0 || x>=N || y<0 ||y>= M)
                                          ret false;
          if ( mat[x][y] == 1 || mat[x][y] == 2)
                                          ret false;
          mat[x][y] = 2;
TC:O(NM)  ret mazeSolver (mat, N, M, x+1, y)
              ||
              mazeSolver (mat, N, M, x, y+1)
              ||
              mazeSolver (mat, N, M, x-1, y)
              ||
              mazeSolver (mat, N, M, x, y-1)
}
```

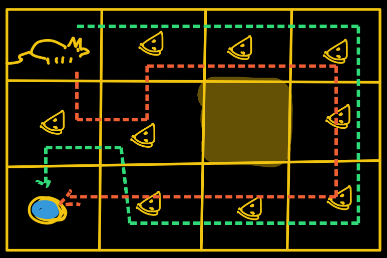
Google

Hard

Rat in a maze

Given the start point of rat, end point, blocked cells, cells filled with cheeze.

Count the no of paths from start to end such that rat can eat all the cheeze present in the maze without stepping on same cell twice in one path.



⇒ 2

start → s_i, s_j

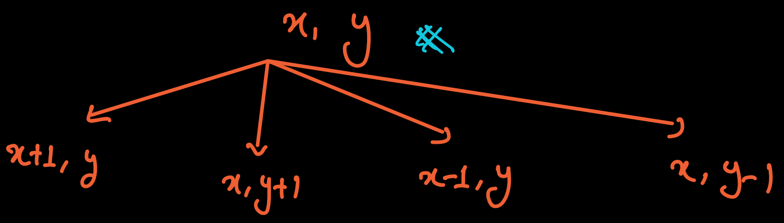
end → e_i, e_j

Cheeze → 0

blocked → 1

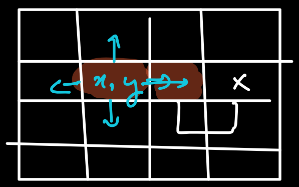
Empty →

Count of Cheeze ⟶ ?   O(NM)

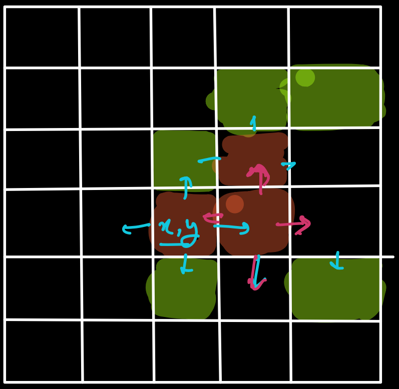


```
int    countPath ( mat , N, M , si , sj , ei , ej , totalChz , curChz ){
      if(si<0 || si >=N || sj <0 || sj >= M)
                                          ret 0;
      if ( mat[si][sj] == 1)
                            ret 0;

      if ( si == ei  && sj == ej ){
            if ( curChz == total Chz){
                   ret 1;
            }
            else
                  ret 0;
      }
         int  temp  =  map[si][sj];
        mat[si] [sj]  =  -1;
       int  ans  =   countPath ( _ _ _ . si+1, sj ,  curChz+1, . . );
                                          +
                     countPath ( _ _ _ . si, sj+1,  curChz+1, . . );
                                          +
                     countPath ( _ _ _ . si-1 , sj ,  curChz+1, . . );
                                          +
                     countPath ( _ _ _ . si, sj-1,  curChz+1, . . );
      mat [si][sj] = temp ;
     ret  ans;
}
```



Only if mat[si][sj]
⇒ 0



fn(x, y+1)

Google
Facebook
MS
Flipkart
Paypal
;
;

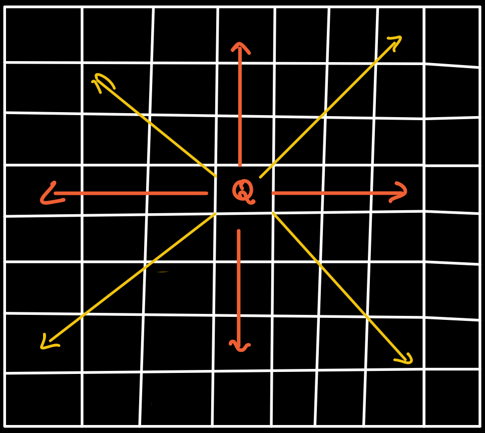
Q N-Queen problem

Given NXN Chess board.

N Queens.

Arrange the Queens in the board such that:

No queen targets any other queen.



N=4

	0	1	2	3
0		Q		
1				Q
2	Q			
3			Q	

	0	1	2	3
0			Q	
1	Q			
2				Q
3		Q		

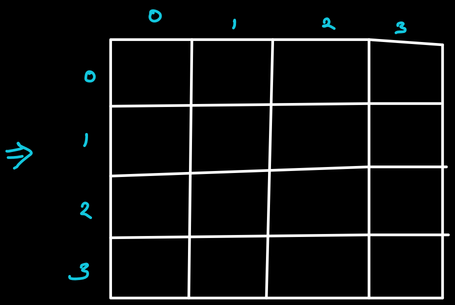
Keep one queen per row,

↳ in every ith row

find a safe col (slot) for the Queen.

if safe col exists → place the Queen

else backtrack

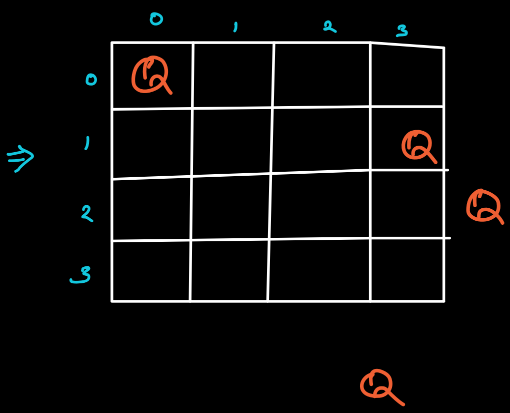


```
void NQueen (mat, N, row) {
        if ( row = = N)
                ret true ;

        for (col=0; col<N ; col++) {
                if( isSafa (mat, row, col)) {
                        mat[row][col] = 1;     HashMap Changes
                        if( NQueen (mat, N, row+1)) {
                                ret true;
                        }

                        mat[row][col] = 0;     Revert HashMap changs
                }
        }

        ret falze;
}
```



r=0 → r=1
c=0   c=2

bool isSafe (mat[][], r , c){ ⇒ O(N)

Return true if no queen is present

~~in row no. r~~

in col no. c → O(1) HashMap<colno, T/F>

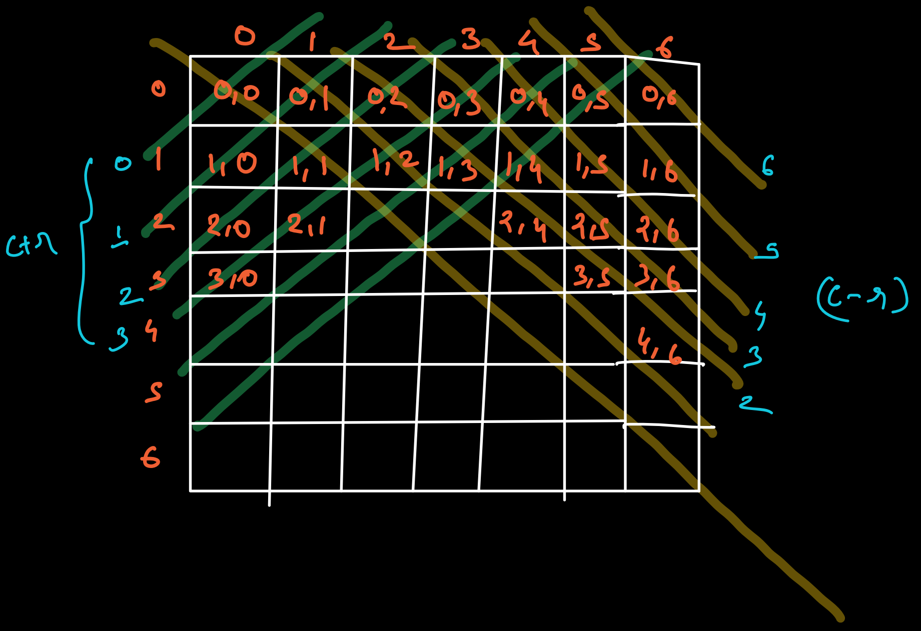
in diagonals.

}

HashMap<c-r, T/F>

HashMap<c+r, T/F>

⇒ O(1)



0 1 2 3 4 5 6
0 1 2 3 4 5 6
0,0 0,1 0,2 0,3 0,4 0,5 0,6
1,0 1,1 1,2 1,3 1,4 1,5 1,6
2,0 2,1 2,4 2,5 2,6
3,0 3,5 3,6
4,6
c+r
0 1 2 3
6 5 4 3 2
(c-r)