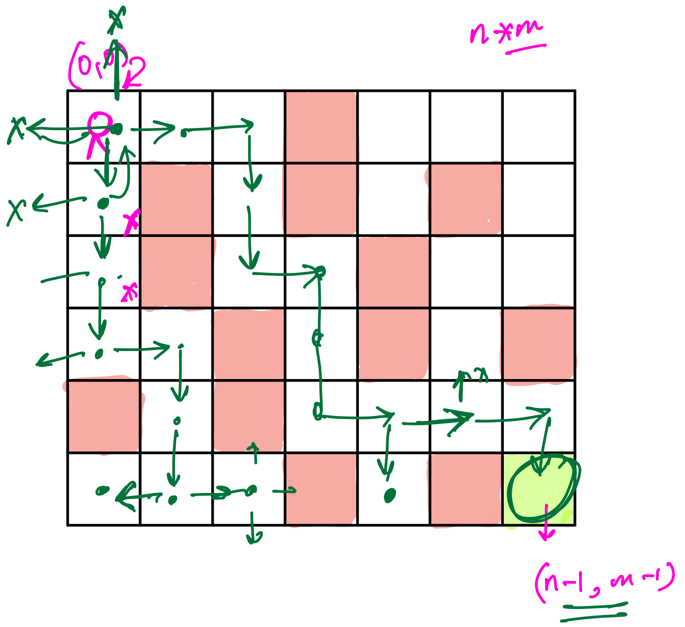
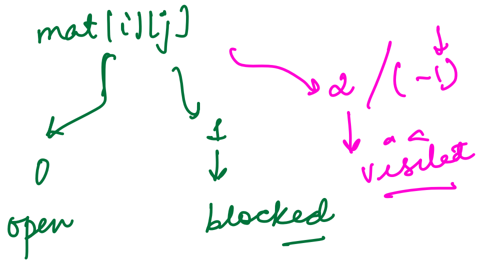
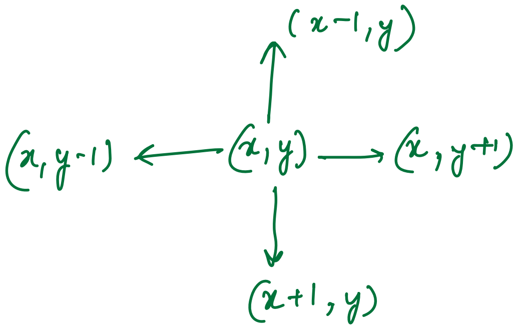
Rat in a maze



if you can reach to your target!

mark cells as visited





parameters

```
bool ratmaze(i, j, mat[][], int n, m)
  {
          if ( i==n-1  && j==m-1) return true;

          if( i<0 || i>=n || j<0 || j>=m)
                                  return false;
          if( mat[i][j]==1 || mat[i][j]==2)
                                  return false;

          mat[i][j] =2;

      return   ratmaze( i-1, j ...) ||
               ratmaze ( i+1, j ... ) ||
               ratmaze ( i, j-1, ...) ||
               ratmaze ( i, j+1, ...)
  }
```

T.C : $O(n*m)$

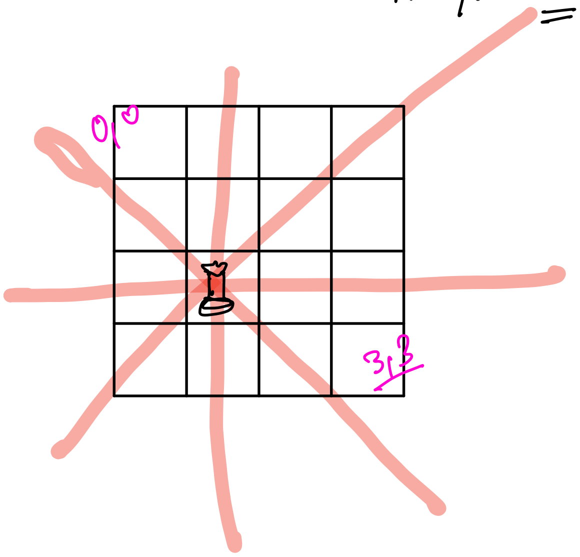
S.C : $O(n*m)$

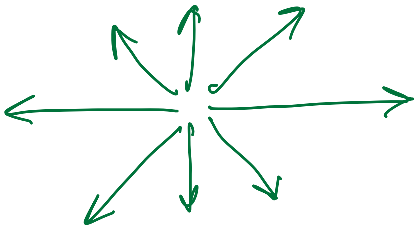
$\cancel{4^{n*m}}$

N queens

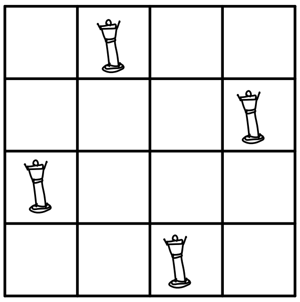
N*N chessboard

N queeens

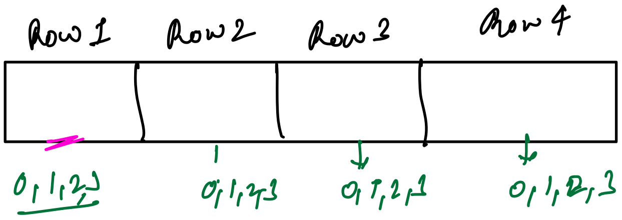


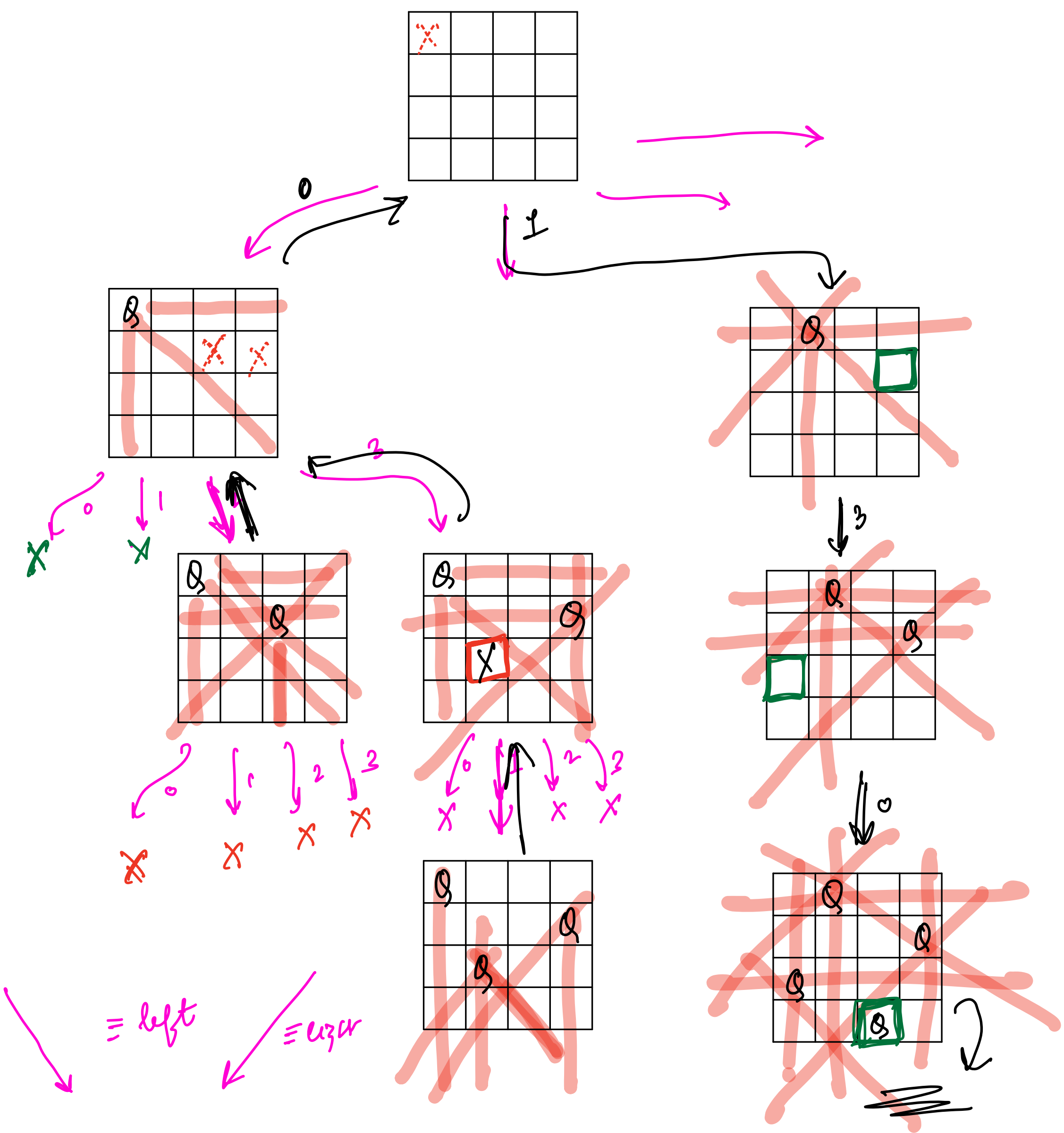


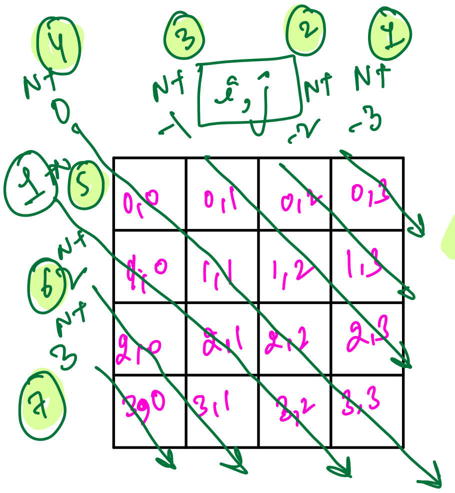
place N queens, such that no 2 queens comes into the way of each other



every row needs one queen







array ↓ initially no column is taken

col[n] = {0};

col[j] = 1;

N+i-j

leftdiagonal[2N] = {0};

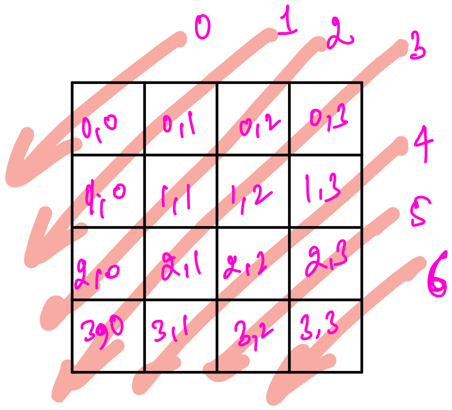
size

If a LD already has a queen

i,j

2,3

2-3 = -1



0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

$i+j$

rightdiagonal [2N]

mat[n][n] ≡ store the config

parameters

```
 Nqueen(int i, int mat[][], int col[], int ld[], int rd[])
{
      if(i==N) { //   got your ans
                        return; }

      for( j=0; j<n; j++)
      {
              // i,j
              if( col[j]==1  || ld[N+i-j]==1
                      || rd[i+j] ==1)
                  continue;

              mat[i][j]=1;
              col[j] =1;
              ld[ N+i-j]= 1;
              rd[ i+j] =1;
              N queens( i+1,  ....);
              mat[i][j]=0;
              col[j] = 0;
              ld[ N+i-j]= 0;
              rd[ i+j] =0;
      }
}
```

0/

	1	2	3	4	5	6	7	8	9
0	5	3	1	2	7	~~4~~6	4	9	8
1	6			1	9	5			
2		9	8					6	
3	8				6				3
4	4			8		3			1
5	7	(5,1)	3		2				6
		6					2	8	
				4	1	9			5
					8			7	9

sudoku 9*9

row ≡ 1-9
column ≡ 1-9
cube ≡ 1-9

go to every cell one by one & try the possibilities

mat[i][j]
1-9
0
empty

index

r = index / 9
c = index % 9

sr = r - r%3
sc = c - c%3

```
sudoku( int index, int mat[][])
{
        if ( index == n*n){ // sudoku solved
                              return }.
            int r= index/n;
                c= index%n;
            if( mat[r][c] !=0) { sudoku( index+1, mat)
                                  return;
                               }

            for( x=1; x<=9; x++)
            {
                    if( check (x, mat, r, c))
                    {
                        mat[r][c]= x;
                        sudoku( index+1, mat);
                        mat[r][c]=0;
                    }
            }
}
```

1) check row if it has x

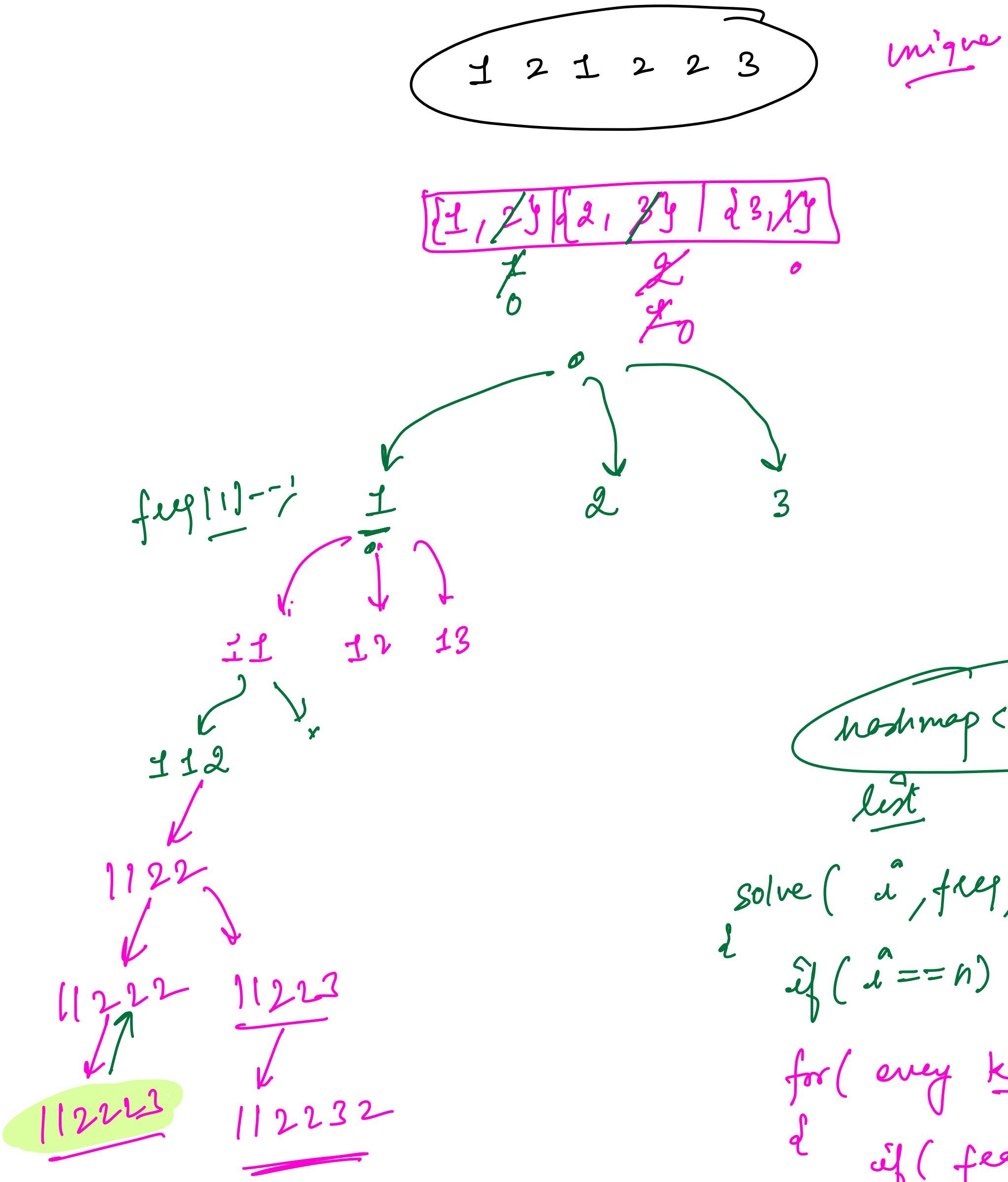
2) check col if it has x

3) check cube

T.C: $O(9^{n*n})$

S.C: $O(n*n)$

- permutation duplicates



```
hashmap<int,int> freq;
list

solve( i, freq, list )
{
    if( i == n) { // ans }

    for( every key in map)
    {
        if( freq[key]>0)
        {
            freq[key]--;
            list.insert(key)
            solve( i+1 ...)
            list.pop_back()
            freq[key]++;
}       }       }
```