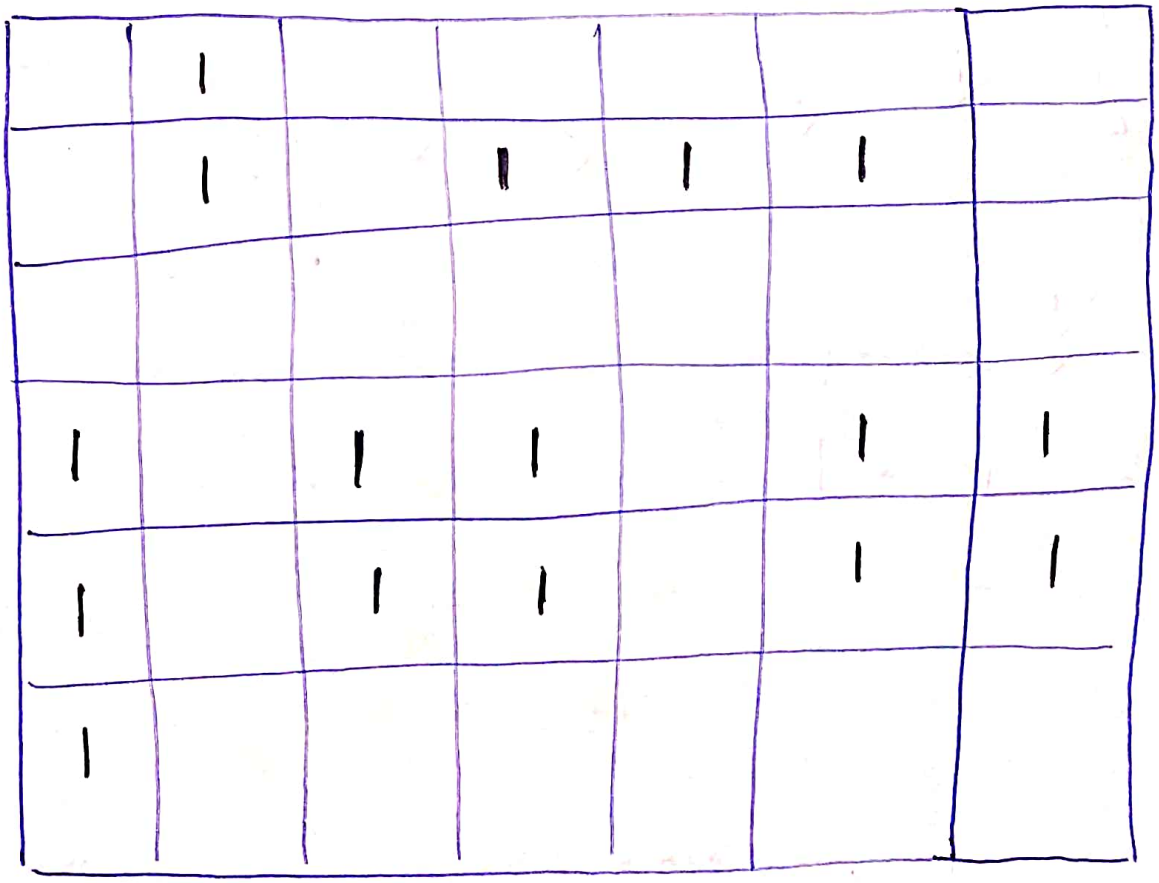
RECURSION - BACKTRACKING

FLOOD FILL

- Hume ek maze di gayi hai | iss maze me har jagah 0 pada hai! Aur kuch boxes of 1 pada hai!
- 0 ka matlab waha se banda chal sakta hai!
- 1 ka matlab waha se banda chal nahi sakta! (obstacle)
- Hum top-left corner se Right-bottom corner jana hai
- hum sirf ek step se top, right, down, left ja sakte hai!
- such that ~~Ko~~ Humko saare rasty print krne hai! Such that koi rasta same cell se Do baar na nikalta ho!
- Calls ka order ← top, left, down, right ↓

Level-option style
Call ← stupid → Base; Smart
smart → Base: Normal



1st TRY

```
p s v m (S[] a){
Scanner s = new Scanner(System.in);
int n = s.nextInt();
int m = ____________;
int [][] arr = new int [n][m];
for(int i=0; i< arr.length ; i++) {
    for (int j= 0; j< arr[0].length; j++) {
        arr [i][j] = s.nextInt();
    }
} Floodfill (arr, 0, 0, "");
}
```

// psf: path so far

```
p s v floodfill (int [][] maze, int row, int col, String psf,
                 boolean [][] visited) {

if (row < 0 || col < 0 || row == maze.length || col == maze[0].length ||
   maze [row][col] == 1) {
    return;
}

if (row == maze.length-1 && col == maze[0].length-1){
    System.out.println(psf);
    return;
}

flood fill(maze, row-1, col, psf + "t");
_____(maze, row, col-1, psf + "l");
_____(maze, row+1, col, psf + "d");
_____(maze, row, col+1, psf + "r");

}
```

Negative Base Case

Positive Base Case

Recursive Calls (STUPID)

BASE CASE

- Row & col negative → row < 0, col < 0
- Row == arr.length
- col == arr[0].length
- maze [row] [col] == 1 (obstacle)

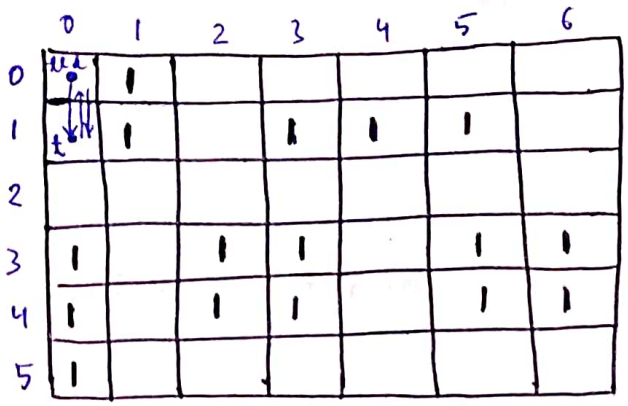
Yeh sabse last me ayega kuki pheli baki condition check hongi unke fail hone pe hi chalana hai isse. Agar yeh pehle chala toh exception ayega

Calls

- Top → maze, row-1, col, psf + t
- left → maze, row, col-1, psf + l
- down → maze, row+1, col, psf + d
- right → maze, row, col+1, psf + r

* ERROR *

= Stack Overflow Error ⟶ KYU?



ROW = 0 COL = 0

= (t) ki call lagi upar se bahar gaye usne wapis bej diya!

= (l) ki call lagi bahar gaye usne wapis bej diya

= (d) ki call lagi khushi khushi niche gaya

0 0	
1 0	t
0 0	tld
1 0	t
0 0	tld

8 C

Aab ROW = 1 COL = 0

= (t) ki call lagi, khushi khushi upar gaya

① ⟶ ② ⟶ ① LOOP

PROBLEM

Problem yeh hai ki hum unhi boxes me dubara jaray hai yaha ek bar ja chukey hai!

Toh humko kuch essa chaiye:

↳ uss same box me dubara nahi jana! jisme ek baar ja chuke hain!

SOLUTION

= Hum ek boolean Array create krengy!

jiska naam visited

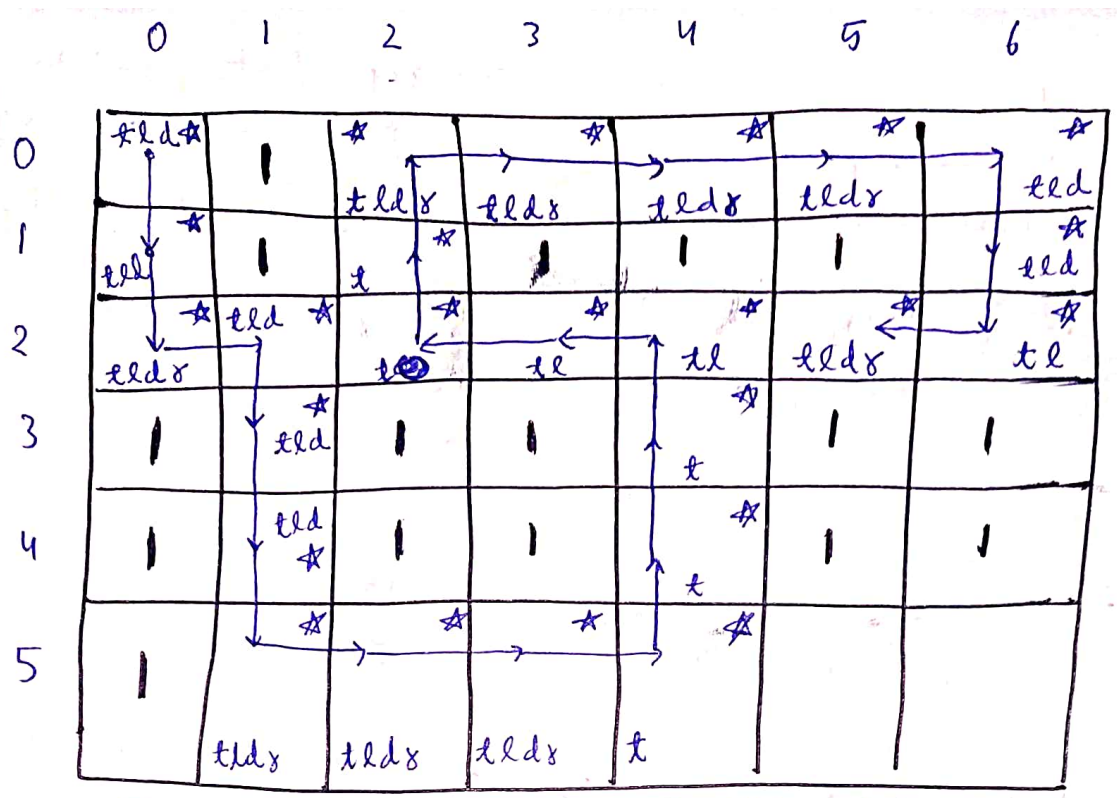
↳ kitna bada hoga? ⟶ utna hi bada jitna bada humara maze array hai!

∴ By Default boolean Array me false hoga!

- Jab bhi ki cell/box me ayengy toh sabse phele uss box/cell ko visited mark krengy! Aur kisi bhi cell me (2nd) time nahi jayengy!

2nd TRY

```
p s v m (s [] a) {
    ____________
    ____________
    ____________
    ____________
    ____________
    ____________
    ____________
    ____________

    boolean [] [] visited = new boolean [n][m];
    floodFill (arr, 0, 0, "", visited);
}

p s v floodFill (____, ____, ____, ____, boolean [][] visited) {
if (row<0 || col<0 || row == maze.length || col = maze[0].length ||
    maze [row] [col] == 1 || visited [row] [col] == true) {
    return;
}
if (_____ && _____) {
    _____
    _____
}
visited [row] [col] = true;
    _____;
    _____;
    _____;
    _____;
}
```



Har box me sabse phele visited ka mark lagega! (☆)

(0,0)
- t → Board ke Bahar: wapis
- l → ——————
- d → Khushi khushi niche

] Atty hi sabse phele ☆ mark

(1,0)
- t → upar visited: wapis
- l → Board ke Bahar: wapis
- d → niche agye!

] Atty hi ☆ mark lagega

(2,0)
- t → upar visited: wapis
- l → Board ke Bahar: wapis
- d → Obstacle: wapis
- r → Right jayega!

] Atty hi ☆ mark lagega

(2,1)
- t → obstacle: wapis
- l → visited: wapis
- d → down jayega!

] ☆

(5,2)
- t Obstacle
- l visited
- d Bahar
- r Right jayega

] wapis ☆

(3,1)
- t visited
- l Obstacle
- d down jayega

] wapis ☆

(5,3)
- t obstacle
- l visited
- d Bahar
- r jayega

] wapis ☆

(4,1)
- t visited
- l obstacle
- d down jayega

] wapis ☆

(5,1)
- t visited
- l obstacle
- d Board se bahar
- r Right jayega

] wapis ☆

(5,4)
(4,4)
(3,4)
→ t jayega ☆

(2,4) (2,3) → (t) obstacle : wapis (l) jayega ★

(2,2) (1,2) → (t) jayega ★

(0,3) (0,4) (0,5) → (t) Bahar (l) visited (d) obstacle (r) jayega!] wapis ★

(2,6) → (t) visited : wapis (l) jayega ★

(2,5) → (t) obstacle (l) visited (d) obstacle (r) visited] wapis ★

(0,2) → (t) Bahar : wapis (l) obstacle (d) visited (r) jayega ★

(0,6) → (t) Bahar (l) visited (l) jayega] wapis ★

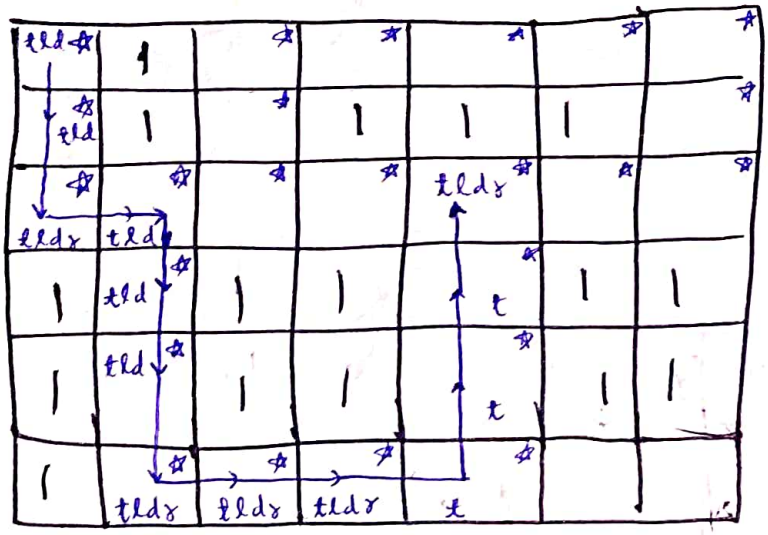
(1,6) → (t) visited (l) obstacle (d) jayega] wapis ★

= Iss stage pe hum path ke end tak poch gaye ki hume forward nahi ja. sakty
∴ We will retreat the path
↳ jiska jiska (tldr) ho jayega vo stack se pop out hoga

= ∴ For retreating, a level of stack keeps poping as we return to the previous level in the stack becoz test case Failed.

= As we keep on doing it,
we reach the position of cell (2,4) < (t) (l)
we tried to move < down, right } marked visited]

∴ No path goes through it.



Problem

we will face problems

↓

Jab hum wapis/ ~~retreat~~ krengy tab

↓

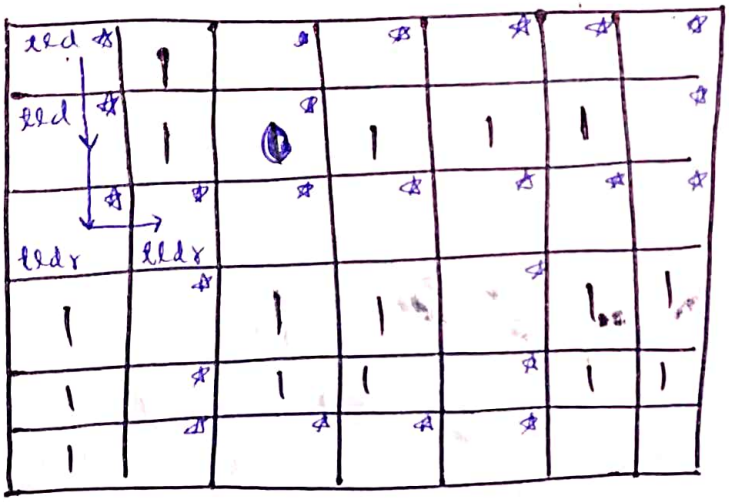
kuki visited mask laga hua hai

↓

new path nahi nikal payengy

↓

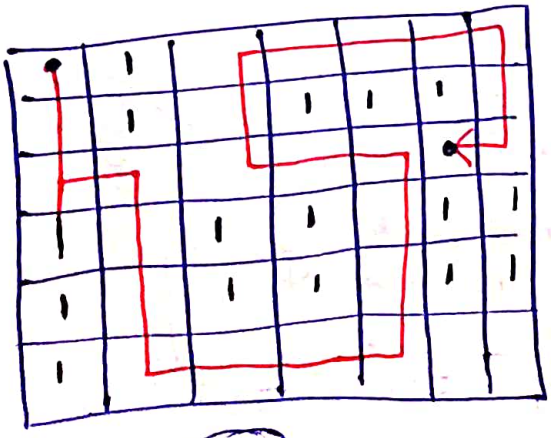
Solution



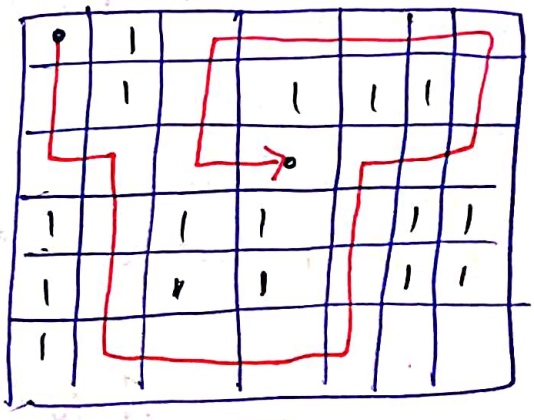
Hume Recursion se wapis aaty waqt visited ko udana hai

↓

Ku? → Hum new paths ko travel krpayengy!



(1)



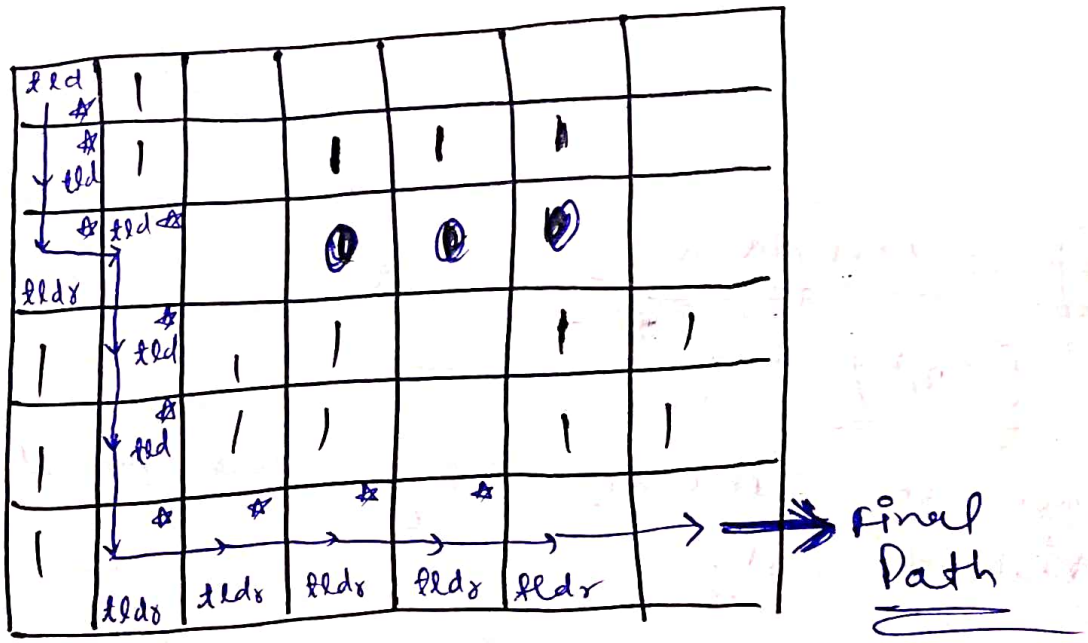
(2)

3RD TRY

```
psvm(s[]a){
}
psvfloodfill(—,—,—,—,—){
if(—||—||—||—||—||—){
}
if(—&&—){
}
visited[row][col] = true;
visited[row][col] = false;
}
```

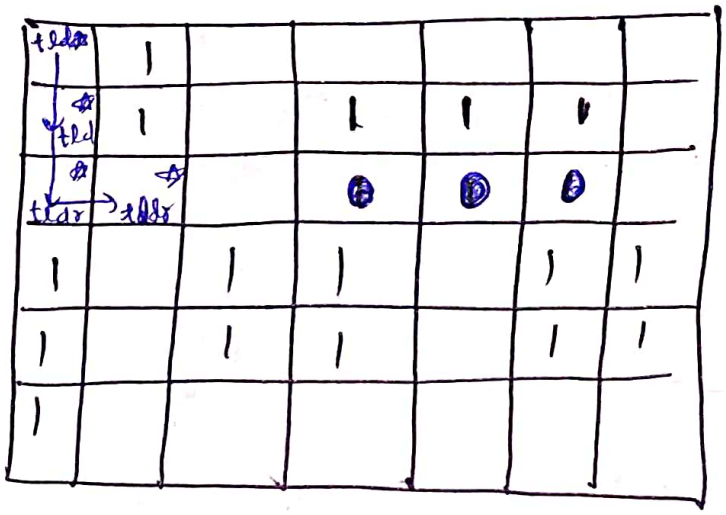
- since after all recursion calls have run completely and returned.
- We mark that cell as false again; ∴ we are free to move on that cell if we want to
- NOW, we retreat, the tick/mark is also removed. see (1) and (2)

- In 2nd try we follow the (1) and after the path retreating and once again marked the visited cells as unvisited, we looked for new path and followed (2).
- Hence, we again retreat,



Hence : First Path : d d r d d d r r r r r

= Again, we start wiping out the cells untill we reach this stage:



= Again, the cells are visited and unvisited multiple times untill we find a new path.

Time Complexity

$$O(4 * n^2) \longrightarrow O(n^2)$$

Because;
Ek cell matrix ka (4) baar chalega
- top
- left
- down
- right

Space Complexity

$O(n^2)$

Since an extra 2D array used for storing "visited" cell.

Space Complexity is Quadratic.

Target Sum Subset

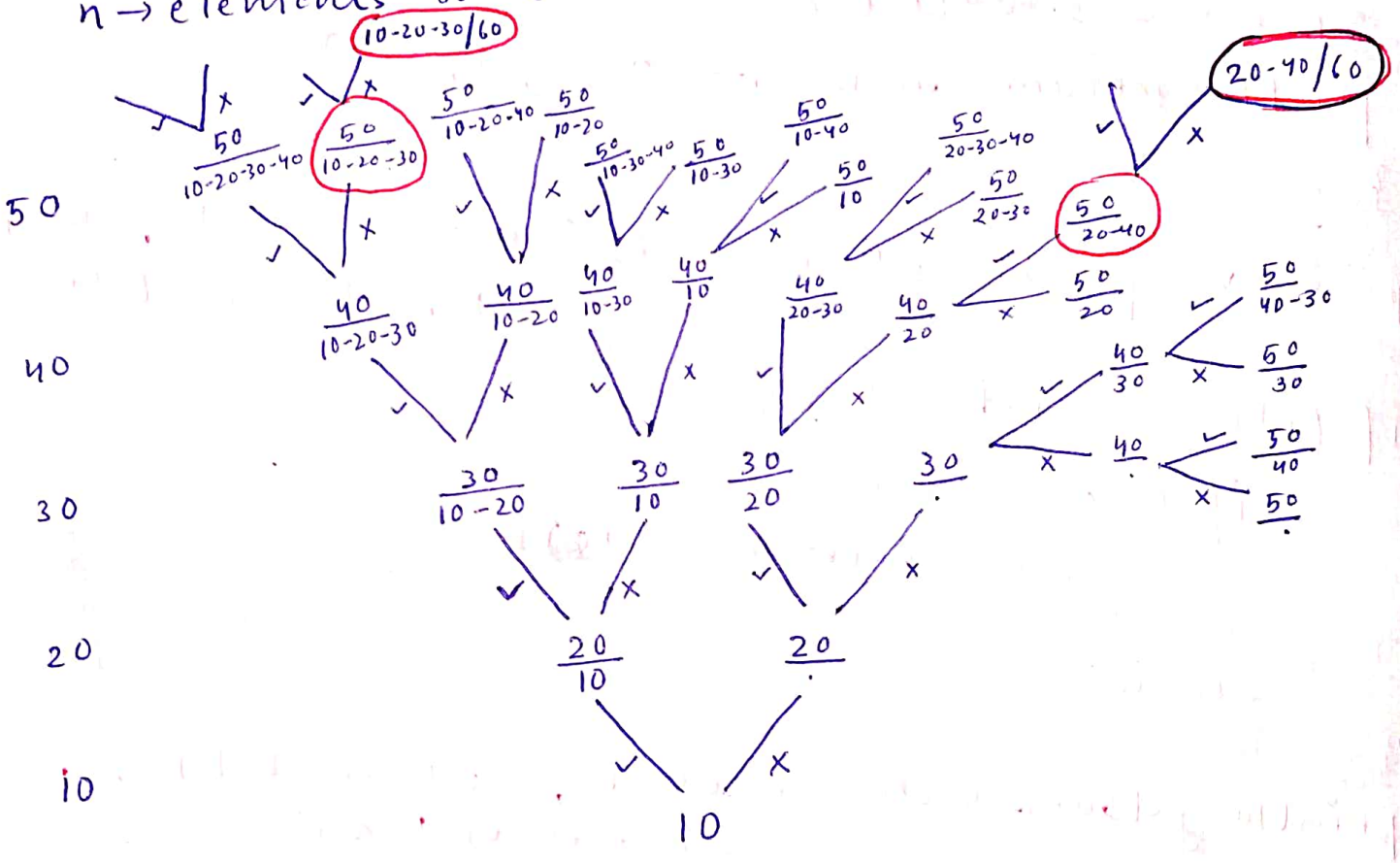
- Humko array mila hai

Target = 60

10	20	30	40	50

- Hume saare subset print krne hai jiska sum 60 ke equal hai

n → elements ∴ 2^4 → subsets



printTargetSumSubsets Function

- Base case
  - if (idx == arr.length)
    - if (sos == targ)
      - return
- Recursive calls
  - choice 1
  - choice 2

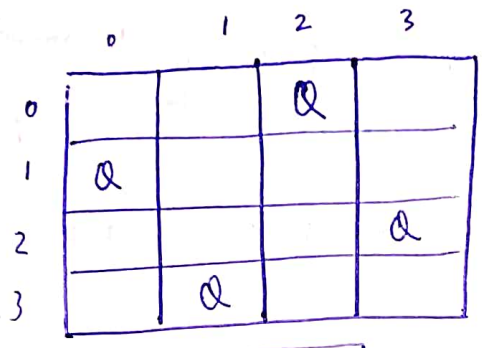
```
p s v m (s[] a) {
 Scanner s = new Scanner (System.in);
 int n = s.nextInt();
 int[] arr = new int[n];
 for (int i=0; i<arr.length; i++) {
    arr[i] = s.nextInt();
 }
 int tar = s.nextInt();
 printTargetSumSubsets(arr, 0, "", 0, tar);
}

p s v printTargetSumSubsets (int[] arr, int idx, string
                             set, int sos, int tar){

if (idx == arr.length) {
    if (sos == tar) {
        System.out.println(set + ".");
    }
    return;
}
printTargetSumSubsets (arr, idx+1, set+arr[idx].+ ",",
                       sos + arr[idx], tar);
printTargetSumSubsets (arr, idx+1, set, sos, tar);
}
```

Time Complexity: $O(2^n)$ → 2 recursive calls

Space Complexity: $O(1)$

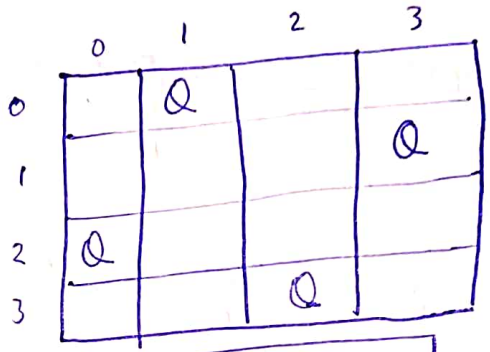
N-Queens

- nxn ka chess board diya hai! (input ⓝ)
- Iss size ke chess board me ⓝ no. of Queens ko place krna hai!
- Such that **no queen kill another queen.**



[0,2, 1,0, 3,1, 2,3]

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

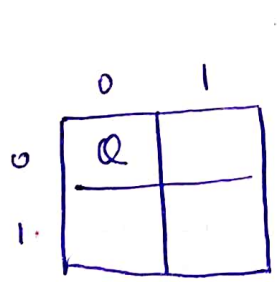


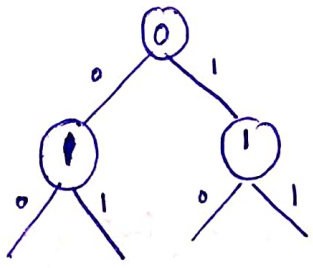
[2,0, 0,1, 3,2, 1,3]

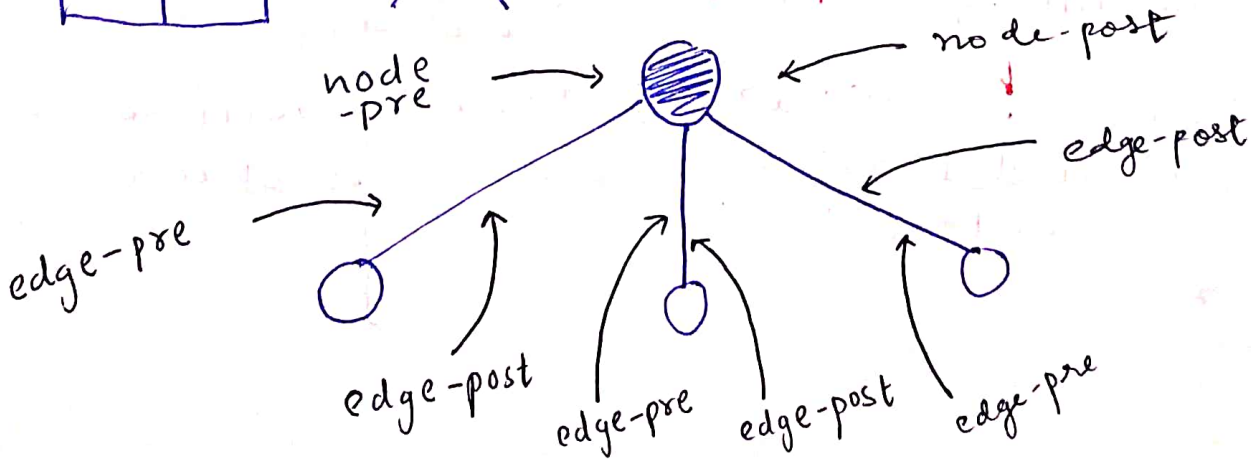
2-0, 0-1, 3-2, 1-3

* ⓝ rows hai aur ⓝ columns hai
* ⓝ Queens place krni hai
* 1 row aur 1 column me 2 Queens baith nahi sakti
* ∴ Toh Har row me 1 Queen ayegi

We will use **Level-Option Style** → Level : Rows, Options : Columns



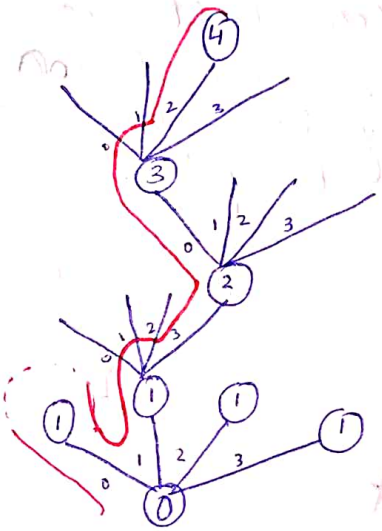


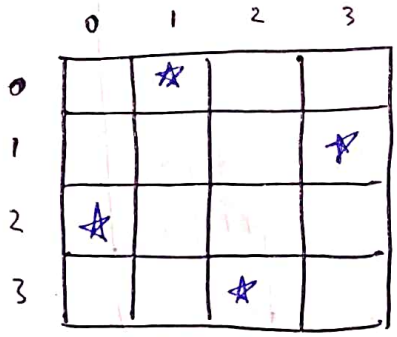


```
p s v m(S [] a){
Scanner s = new Scanner(System.in);
int n = s.nextInt();
int [][] chess = new int [n][n];
printNQueens(chess, "", 0);
}

p s v printNQueens(int[][] chess, String qsf, int row){
// Levels generally humare parameters ke roop me
   atty hai!
// Options generally loops and calls me nazar atty
   hai!
   if (row == chess.length){                    } BASE CASE
      System.out.println(qsf);                  }
      return;
   }
                  (qsf + ".");

for(int col = 0; col < chess.length; col++) {   } choosing columns for given row (OPTIONS)
   if(isItaSafePlaceForTheQueen(chess, row, col) == true){
      chess[row][col] = 1;   -> Edge-pre Statement
      printNQueens(chess, qsf + row + "-" + col + ", ", row+1);
      chess[row][col] = 0;   -> Edge-post Statement
   }
}

p s boolean isItaSafePlaceForTheQueen(int[][] chess, int row,
                                      int col){
for(int i = row-1, j = col; i >= 0; i++){       } -> vertical check
   if(chess[i][j] == 1){
      return false;
   }
}
```

```
for(int i=row-1, j=col-1; i>=0 && j>=0; i--,j--){
    if (chess[i][j]==1){
        return false;
    }
}
```
} Diagonal check

```
for (int i=row-1, j=col+1; i>=0 && j<chess.length; i--,j++)
{   if (chess[i][j]==1){
        return false;
    }
}
```





Time complexity : $O(n!)$

HOW ?

Assume,

Run-time for n rows = $T(n)$

(n-1) rows = $T(n-1)$

Since we are making at max n calls,

$$T(n) = n \times T(n-1)$$

$[\because T(n-1) = (n-1) \times T(n-2) + 1] \times n$

$[T(n-2) = (n-2) \times T(n-3) + 2] \times n(n-1)$

$[T(1) = 1 \times T(0) + (n-1)] \cdot \{n(n-1)(n-2)\cdots 1\}$

$\therefore T(n) = \{n(n-1)\cdots 1\} + \{n + (n(n-1)) + (n(n-1)(n-2)) + \cdots (n(n-1)\cdots 1)\}$

$$T(n) = n!\,T(0) + \left\{\frac{n!}{1} + \frac{n!}{1\cdot 2} + \frac{n!}{1\cdot 2\cdot 3} + \cdots \frac{n!}{1\cdot 2\cdot 3\cdots n}\right\}$$

$$T(n) = n!\left\{1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}\right\}$$

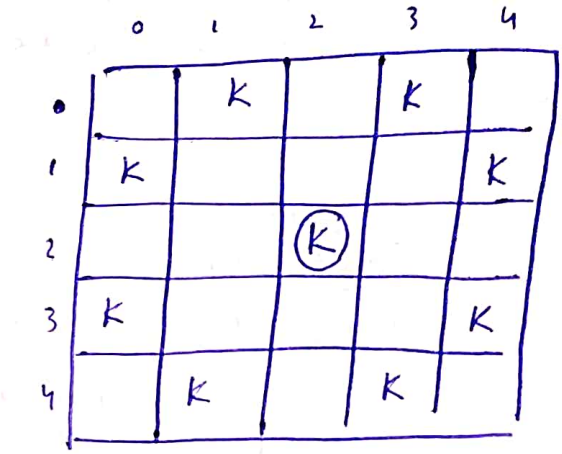
↳ Ignoring this as $n \to \infty$, $\frac{1}{n!} \to 0$

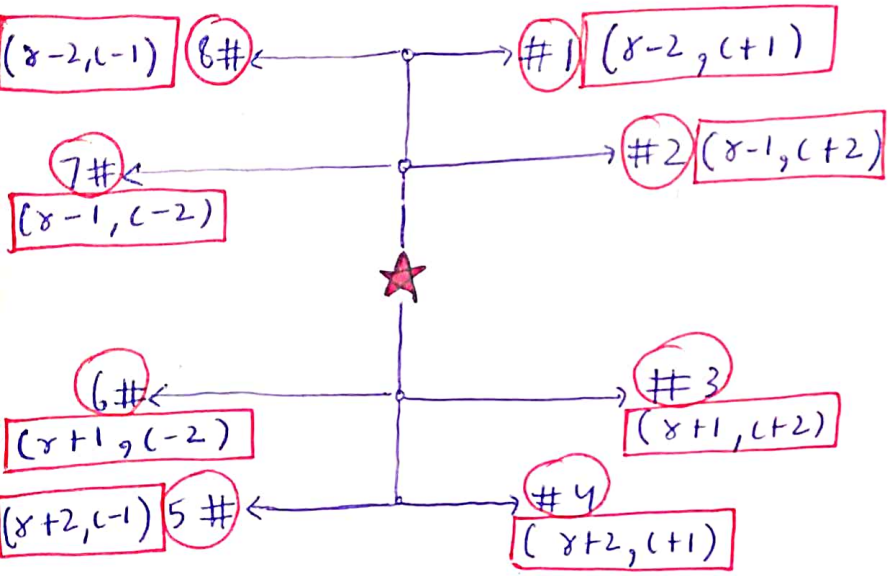
$$T(n) = n!$$

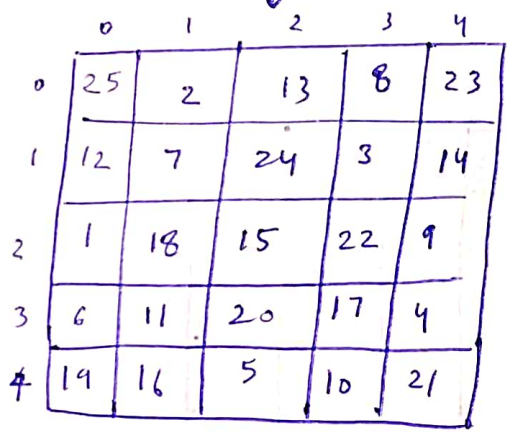
Space complexity : $O(1)$

Knights Tour

- Hume ek chess board mila hai 5 x 5 size ka
- Hume ek box ya cell dengy!
- Iss cell se knights apne moves chalega!
- Hume (K) ko move krna hai
- Iss trah se move krna hai ki (K) issi ke ander chalta rahey aur 25 (Total) saare boxes ko visit krdey bina kisi box ko 2 baar visit krey!








```
public static void main (String [] args){
  Scanner s = new Scanner (System.in);
    int n = s.nextInt();
    int r = s.nextInt();
    int c = s.nextInt();

    int[][] chess = new Int [n][n];
    printKnightsTour (chess, r, c, 1);

}
public static void printKnightsTour (int[][] chess, int r, int c,
                                     int move){
if (r<0 || c<0 || r>=chess.length || c>=chess.length || chess[r][c] >0)
  {  return;

  }
```

```
else if (move == chess.length * chess.length) {
    chess[r][c] = move;
    displayBoard(chess);
    chess[r][c] = 0;
    return;
}

chess[r][c] = move;
printKnightsTour(chess, r-2, c+1, move+1);
_____(chess, r-1, c+2, move+1);
_____(chess, r+1, c+2, move+1);
_____(chess, r+2, c+1, move+1);
_____(chess, r+2, c-1, move+1);
_____(chess, r+1, c-2, move+1);
_____(chess, r-1, c-2, move+1);
_____(chess, r-2, c-1, move+1);

chess[r][c] = 0;

}
public static void displayBoard(int[][] chess) {
  for(int i=0; i<chess.length; i++) {
      for(int j=0; j<chess[0].length; j++) {
          System.out.println(chess[i][j] + " ");
      }
      System.out.println();
  }
System.out.println();

}
```

Time Complexity: $O(8^{n^2})$ ← each cell has 8 decisions from each of the n^2 cells

Space Complexity: $O(n^2)$