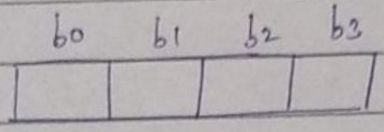
BACKTRACKING

Queen Permutation - 1D

explicit undo

n boxes, 2 queens



$^4P_2$

$^nP_2$

2 queens → q_0, q_1

$^4C_2 \cdot 2! = {}^4P_2 = (12)$ ways.

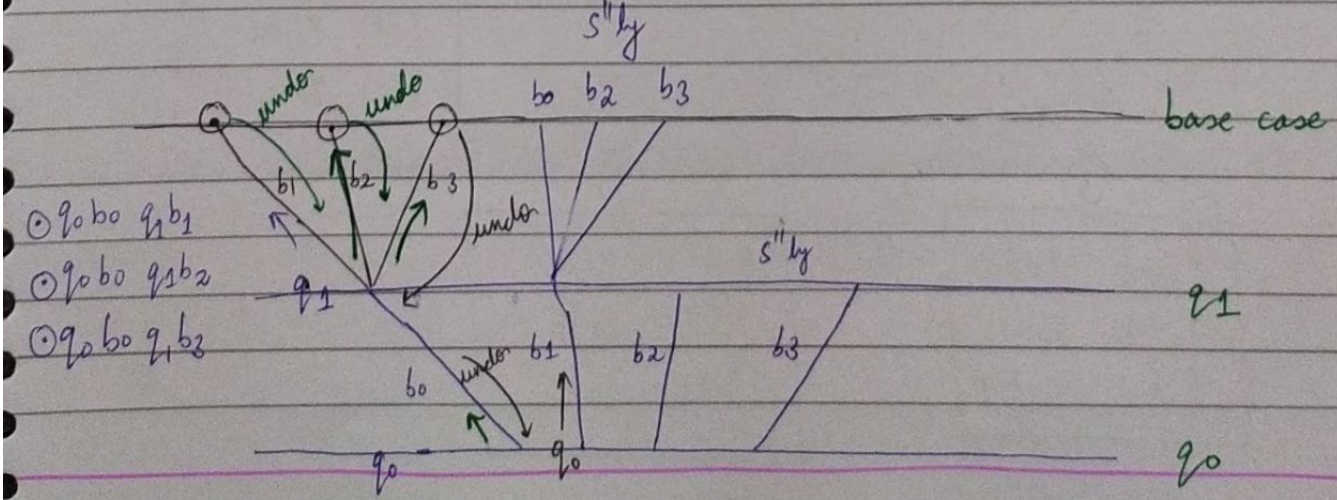
Let's explore these 12 ways :-

$^4C_2 = 6$

$b_0\ b_1$	→	2	$q_0 q_1$, $q_1 q_0$
$b_0\ b_2$	→	2	s"ly as $b_0 b_1$ case
$b_0\ b_3$	→	2	
$b_1\ b_2$	→	2	
$b_1\ b_3$	→	2	
$b_2\ b_3$	→	2	

$6 \times 2 = (12)$

RT :-



* So what we did ⇒

① q_0 ko b_0 me baithya

② q_1 ko baithane ke 3 options ← b_1, b_2, b_3

③ q_1 ko b_1 me baithaya ⟶ q_0b_0 q_1b_1

④ q_1 ko b_1 se wapas nikaala (Backtrack karwaya)

⑤ q_1 ko b_2 me baithaya ⟶ q_0b_0 q_1b_2

⑥ q_1 ko b_2 se wapas nikala ⟶ (Backtrack karaya)

⑦ q_1 ko b_3 me baithaya ⟶ q_0b_0 q_1b_3

⑧ Now, q_0 ko Backtrack karaya

⑨ q_0 ko baithane ke 2 options ← b_1, b_2, b_3

⑩ q_0 ko b_1 me baithaya

⑪ q_1 ko baithane ke 3 options ← b_0, b_2, b_3

⋮

& so on...

③

q_0b_1	q_1b_0
q_0b_1	q_1b_2
q_0b_1	q_1b_3

S"ly

③

q_0b_2	q_1b_0
q_0b_2	q_1b_1
q_0b_2	q_1b_3

③

q_0b_3	q_1b_0
q_0b_3	q_1b_1
q_0b_3	q_1b_2

⇒ 3 × 4 = 12

box

* boolean[] board ;

tq → total queens (2)

qpsf → kitne queens place kara diye hain so far

ans → print jo karana hai usko ans me store karunge

* q0 → ke paas kitne options hain ? → Total no. of boxes

Toh kyun na main iske liye loop laga dun, q0 ko ek ek karke boxes me baitha dun.

* Acha q1 ke boxes tabhi explore karenge jab

* queen baith gyi → true
otherwise → false

* Ye make sure karna hamara kaam hai ki ek hi box me dono queens ko na baitha dein

Toh jahan box false hoga → wahan pe hi baitha sakte

* BC if (qpsf == tq) {
// print ans ;
then return
}

* Step undo karne ke liye → false kedo box ki array me kya kiya jaaye

if

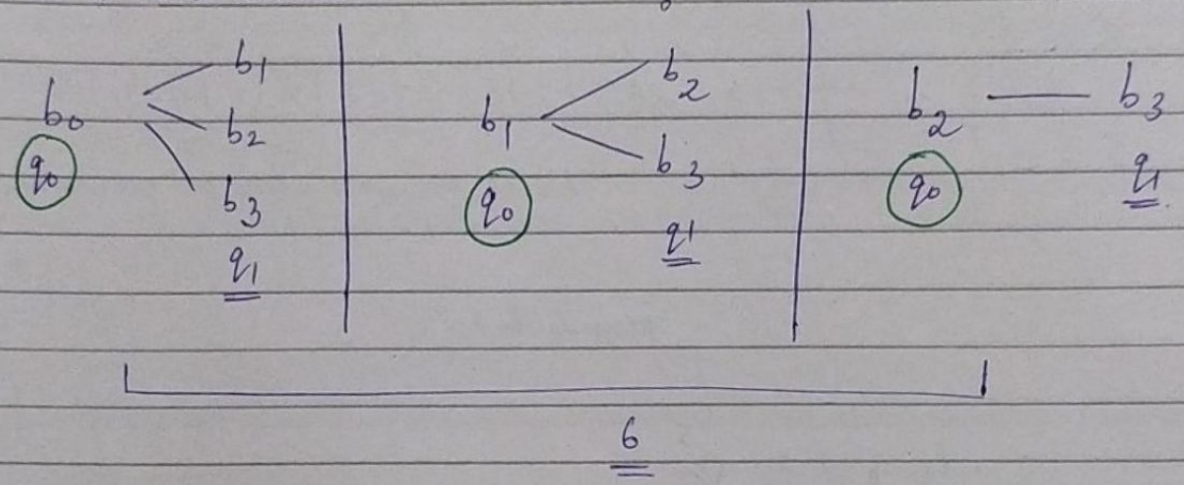
Code :-

```
static int count = 0;

p s v m ( ) {
    QueensPermutation (new boolean[4], 0, 2, "");
}

p s v QueensPermutation (boolean[] boxes, int qpsf, int tq,
                         String ans) {

    // Base Case
    if (qpsf == tq) { // Jab 2 queens ko baitha diya, tab count ko badhayenge aur ans print karayenge
        count++;
        sysO (count + "." + ans);
        return;
    }

    for (int i = 0; i < boxes.length; i++) {
        if (boxes[i] == false) { // to ensure ki ek box me 2 queens na baithe ho
            boxes[i] = true;
            // Recursive call, ek queen ko baithaane ke baad
            QueensPermutation (boxes, qpsf + 1, tq,
                ans + "q" + qpsf + "b" + i + "");
            boxes[i] = false; // undo operation (Backtrack)
        }
    }
}
```

Queen Combination

$^4C_2$ * Here it doesn't matter, q_0 q_1 se pehle hai ya baad me.
* $12/2 = (6)$ Ans.

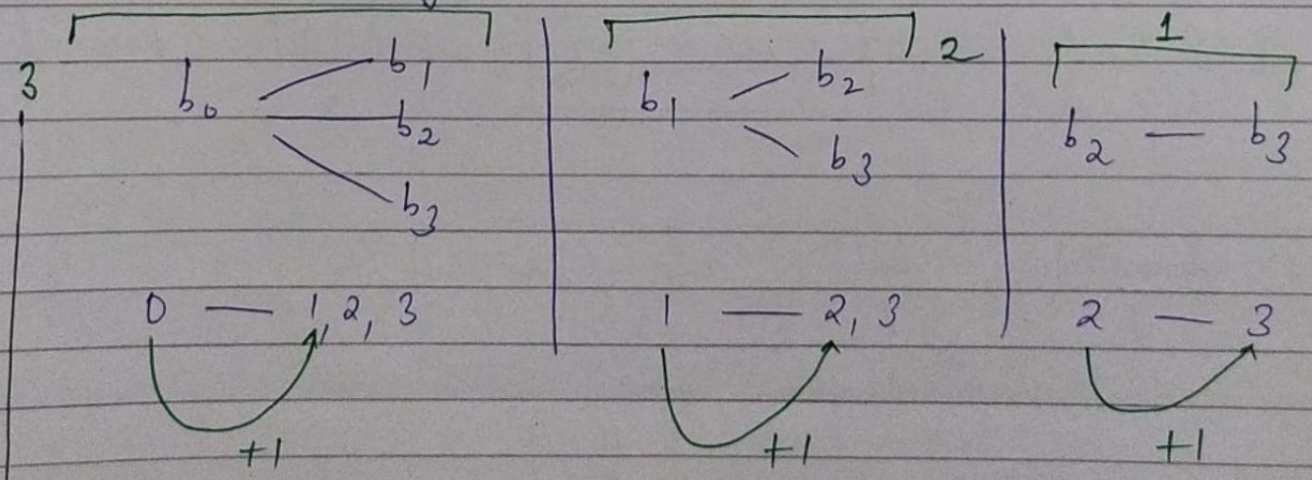
⇒ Toh kaise discard karun calls ko?



6

⇒ ek parameter use karenge → lastBoxUsed

⇒ An imp. thing is: ki ab for loop ko (lastBoxUsed +1) se start karna hoga. Aisa kyun?



⇒ When calling the fx^n in main, lastBoxUsed ko -1 se start karenge, because agar 0 se start kiya toh 0+1 = 1 se start hoga, toh pehle 3 calls challenge hi nhi.

Coin Change → Implicit undo

→ infinite supply

Denomination : {2, 3, 5, 6}

amount : 10

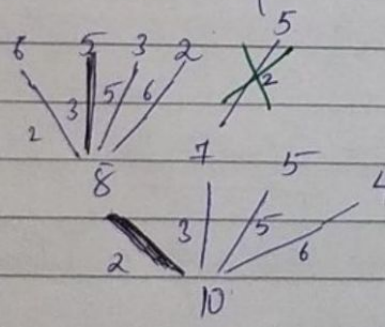
Combinations

1. 2 2 2 2 2
2. 2 2 3 3
3. 2 2 6
4. 2 3 5
5. 5 5

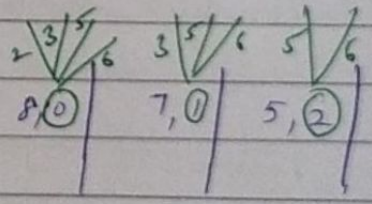
→ ~~253~~ | ~~325~~ | ~~352~~ | ~~523~~ | ~~532~~

We're not interested in permutations.

→ permutation hai



5 "by" for all



2	3	5	6
0	1	2	3

lastDenomIdx

denom[ ] , amt , ans , lastDenomIdx

↓ ↓

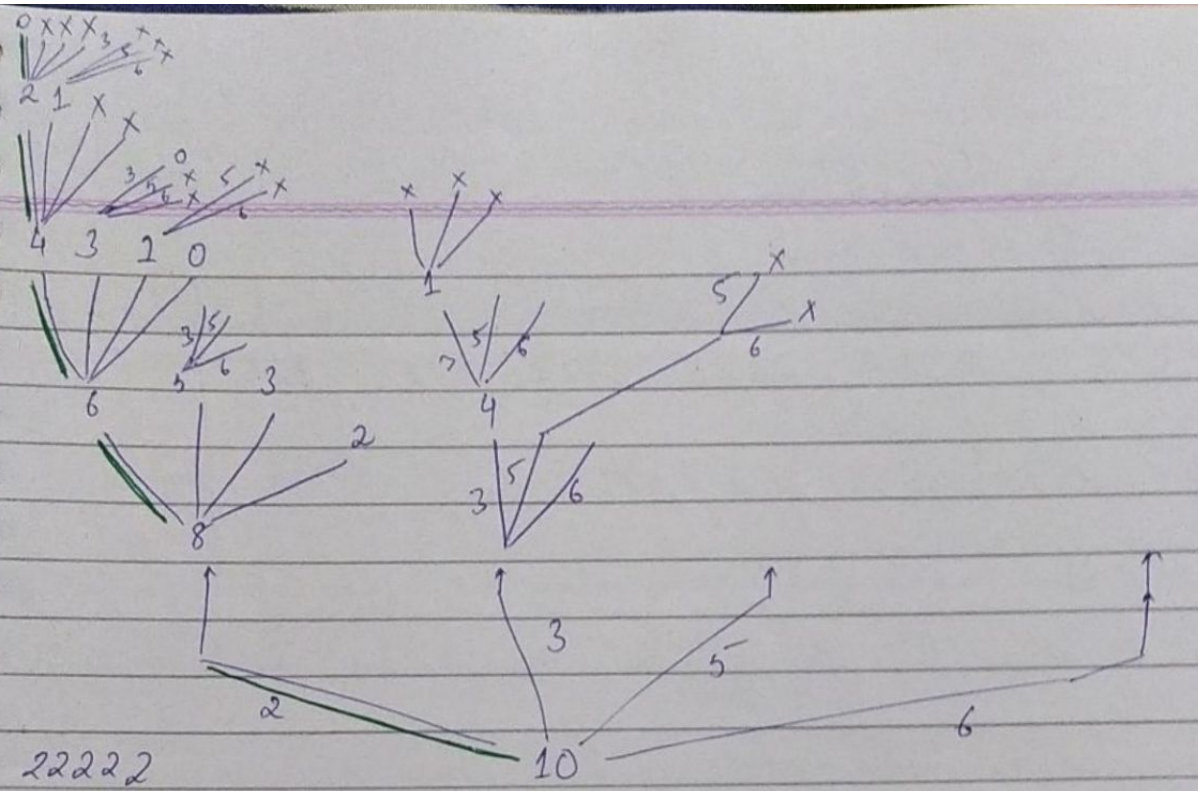
2,3,5,6 10

10 - 2

↑ ↓

amt - denom[i]

Code → github



22222
2233
226
235
55

Code :-

```
p s v CoinChange (int[] denom, int amount, Stng ans,
                  int lastDenomIdx) {

    if (amount == 0) {
        syso (ans);
        return;
    }

    for (int i = lastDenomIdx ; i < denom.length ; i++) {
        if (amount >= denom[i]) {
            CoinChange (denom, amount - denom[i],
                        ans + denom[i], i);
        }
    }
}
```

jaise 10 → 7 agar 3 wala denomination use kiya, toh uska index (1) tha, aur i & uske baad se hi aage ke calls chahiye

Coin Change Permutation

2 2 2 2 2	→ 1
2 2 3 3, 2332, 2323, 3322, 3223, 3232	→ 6
226, 262, 622	→ 3
235, 253, 325, 352, 523, 532	→ 6
55	→ 1
	17

* Hamne kya kiya tha last ques me permutations bachane ke liye?

LastDenomIndex → use kiya tha

* LastDenomIdx kya track rakhta tha → taaki aage wala level uss index se aur uske aage ka loop lagaye

* LastDenomIdx hatado → itna sa hi change hai code me

* count bhi rakhlei hain → to keep track of the answers.

An imp note:

Hamare paas 2 options hote hain :—

1) Ya toh recursive call lagane se pehle check kelo ki amount negative nhi ho → which we did in prev code

2) Ya ek aur cheez kar sakte ho → ek negative base case lagado upar for loop ke

```
if (amount < 0) {
    return;
}
```

* Dono options me se koi bhi use kar sakte ho, totally upto you.

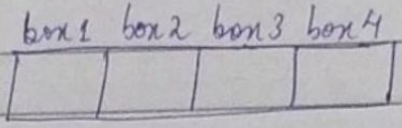
Toh iss case me, we don't need rec call ko check karna like in prev. ques.

Queen Combination

Queens = 2

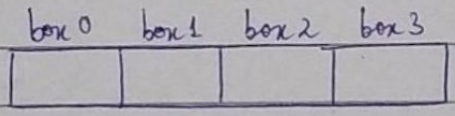
Method : 2

board :

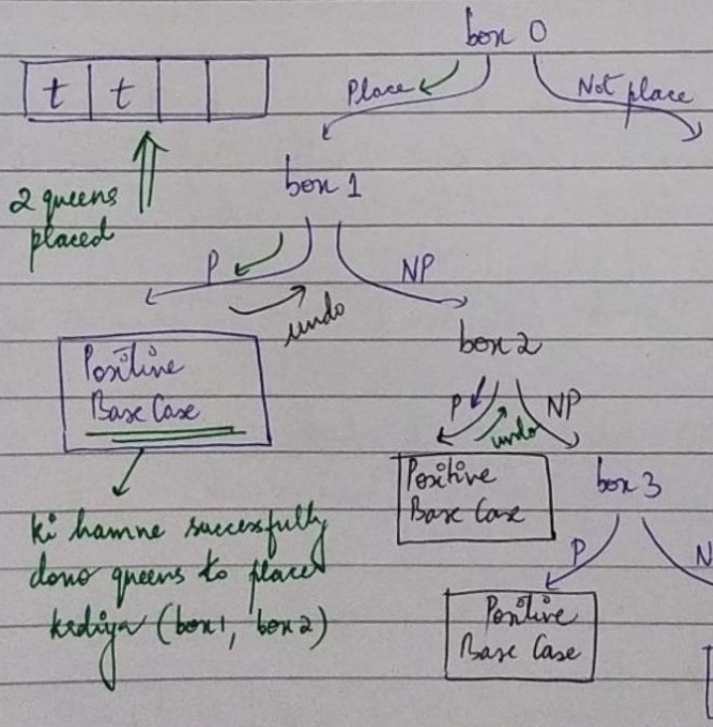


Queen Respect	Box Respect
⇒ Pehle ham ye dekh rhe the ki queen ko b0 b1 b2 b3 me kisme baithayein	⇒ Yahan hain box ke respect me dekh rhe hain, ki bo me Queen place ho sakti hai ya nahi

board



(Box respect)



* Ab we will undo, box 1 se undo kiya (Box 1 pe queen ko place hi mat karo)
Ham NP me chale jayenge

* Then Box 1 → NP ⟹ Box 2 P ∴ q_0 → box 0
q_1 → box 2

* Then q_1 ko undo karo box 2 se, aur box 3 me place krdo

* Now $q_0 \longrightarrow$ box 1
$q_1 \longrightarrow$ box 3

Abhi tak hamne kya kiya ⇒ placed 2 queens (q_0, q_1)

b_0	b_1
b_1	b_2
b_2	b_3

* Ab we will undo & box 3 → NP → box 4

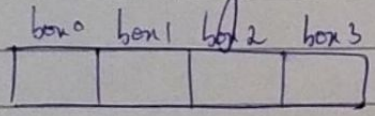
↓

Negative Base Case
(this box does not exist)

* Ab fir undo karenge, aur undo karte karte box 0 pe pahunch jayenge. To bhi answers hamne explore kiye, with saare box 0 one queen ko place karke liye

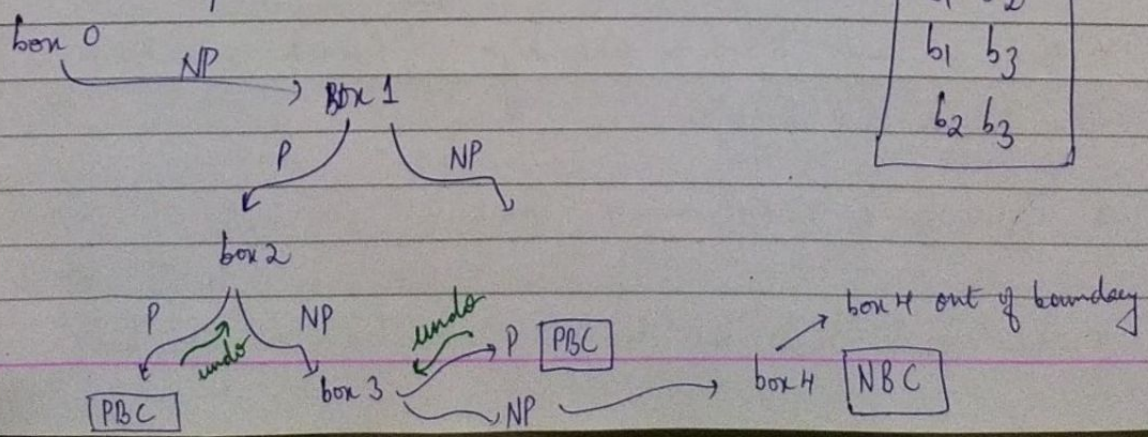
* Jo branch abhi explore ki → woh box 0 → P karne se bani

* Acha undo kaise karenge → box 0 se t hata denge



* Ab we will explore box 0 ki NP call

b_1	b_2
b_1	b_3
b_2	b_3



Queen kaunse col me baithi hai

Code :-

```
p s v QueenCombinationBoxRespect(boolean[] board, int col, int tq,
                                 int qpsf, int tq, string ans){

// positive base case
if(qpsf == tq){
    syso(ans);
    return;
}

// negative base case
if(col == board.length){
    return;
}

// place
board[col] = true;
QueenCombinationBoxRespect(board, col+1, qpsf+1, tq, ans + "b" + col);
board[col] = false    // undo

// not place
QueenCombinationBoxRespect(board, col+1, qpsf, tq, ans);
```

* Imp. Note :-

Agar hame nbc ko pbc se upar likh dein, toh galat o/p (colburn o/p) aayega

Khud socho kyun aisa hai? Suppose $b_2 b_3$ me rakhdiya queens ko, toh b_3 me queen ko place krke agle ki rec call laga diya, toh ek naya frame create hua, toh if pbc is written first $b_2 b_3$ print ho jaayega, lekin agar nbc ko pehle likha, toh col == board.l execute ho jaayega & box 4 ho jaayega & ham return kr jaayenge bina ans print karaye ($b_2 b_3$ print nhi hoga)

Coin Change Coin Respect

Coin Change

user respect

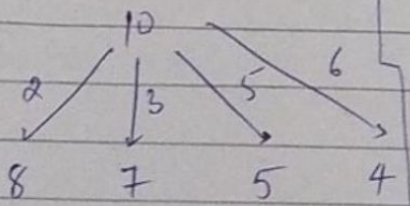
(ki user ke paas option hai with 2,3,5,6 select kar sakta hai)

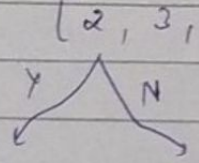
coins respect

★ coins : infinite supply

[2,3,5,6] amt = 10

* Ish 2 ke paas kya option h ki 10 amt banane me 2 contribute karega ya nhi karega

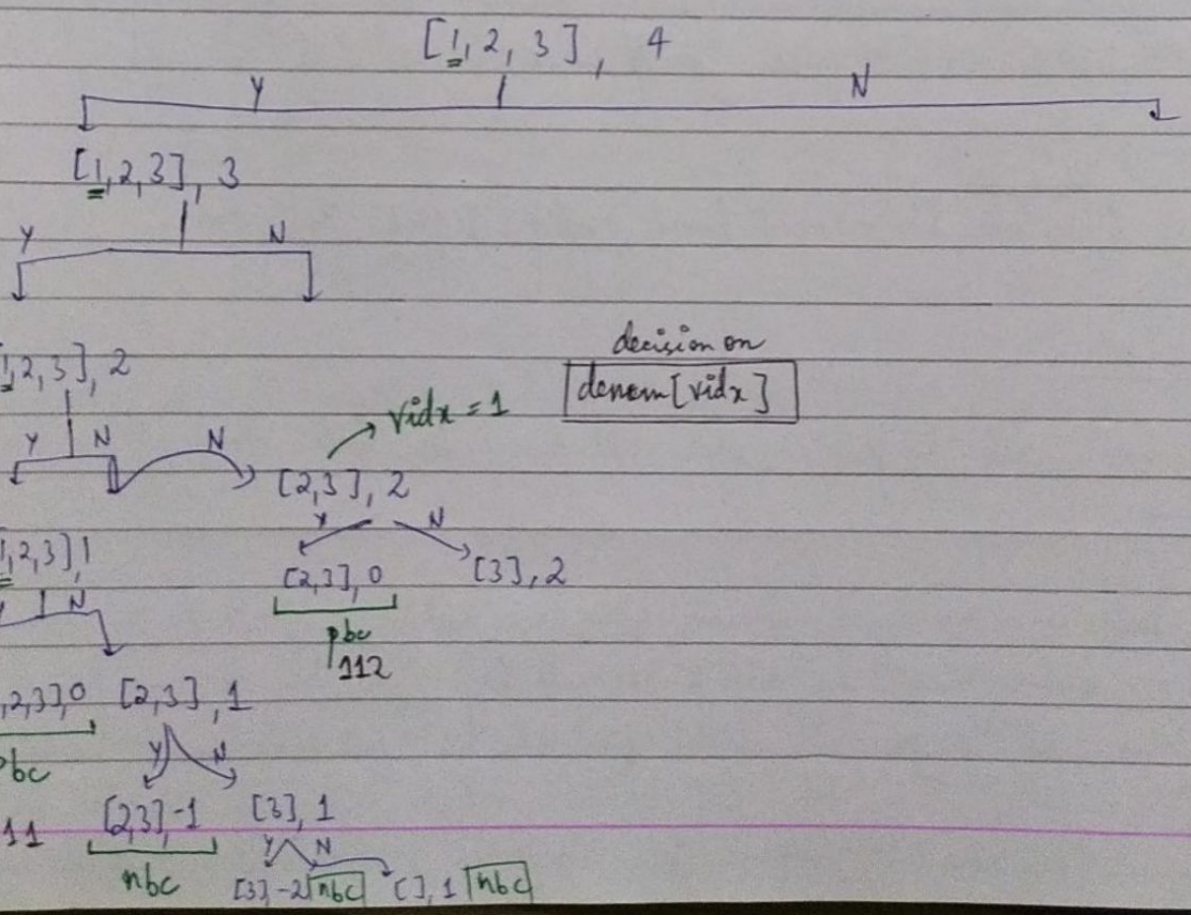




[2,3,5,6], 8 [3,5,6], 10

Eg: [1, 2, 3], 4

⇒ o/p : 1111, 22, 13, 112



decision on denom[vidx]

0 1 2

denom : [1, 2, 3]

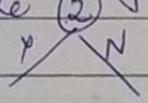
[1, 2, 3] → vidx = 0

[2, 3] → vidx = 1

* array denom wali hamesha [1, 2, 3] hi paas hogi, lekin jab hame eg. [2, 3] chahiye hoga toh with ham vidx se dekhenge

* denom [vidx] → denom [1] → (2)

* Aisa kyun kerna pada array ko virtually break? Ham jab RT bana rhe the, tab ham array ka 1st index ka contribution consider kar rahe the (2)



Obviously ham [2, 3] array pass nhi kr rhe honge, ham puri ki puri array [1, 2, 3] hi pass krenge; toh isliye we need vidx to virtually break the array

* (1) Virtually break : vidx
(2) Decision : denom [vidx]

Code :-

// Discussion of Base Case

* Acha we have 2 BC → pbc, nbc → 1 type, 2 type

* 2 types of nbc → 1. array blank hona, vidx = 3
2. amount -ve ho jaaye

```
p s v CoinChangeCoinRespect(int[] denom, int vidx, int amount,
                                        String ans) {
    // Base Case
    // yes

    CoinChangeCoinRespect(denom, vidx, amount - denom[vidx],
                                        ans + denom[vidx]);

    // no

    CoinChangeCoinRespect(denom, vidx+1, amount, ans);

}

    // Base Case

    // pbc  (positive base case)
    if(amount == 0) {
        syso(ans);
        return;
    }

    // nbc  (negative base case)
    if(amount < 0 || vidx == denom.length) {
        return;
    }
```

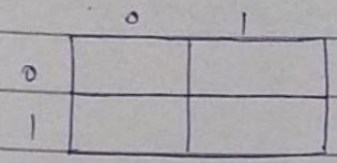
* Implicit undo

⇒ rec. call khatam hone ke baad ans wapas wahi ho jayeg

Queen Combination 2D

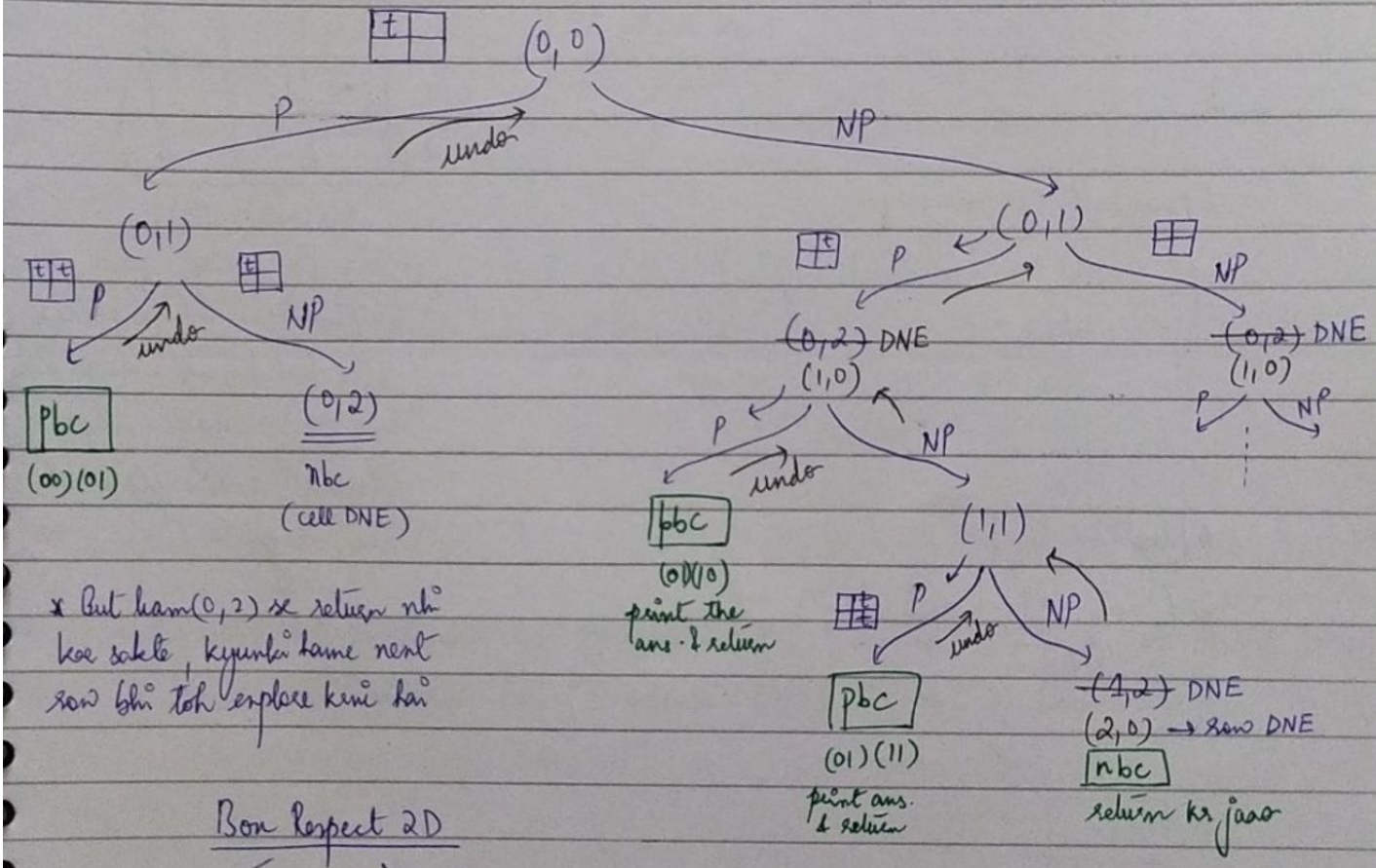
2D Board

→ row, col
→ 2 Queens place : Combination
→ Box Respect



Suppose.

Recursion Tree

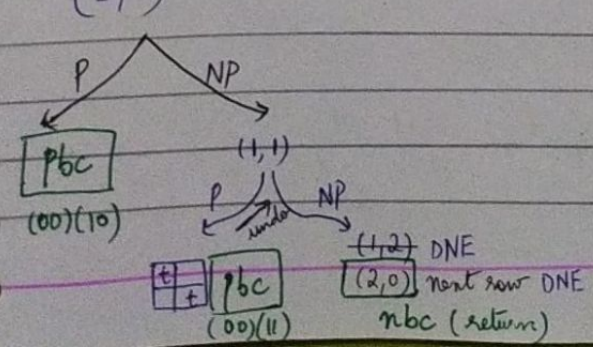


* But ham (0,2) se return nhi kar sakte, kyunki hame next row bhi toh explore krni hai

Box Respect 2D

Change variables manually

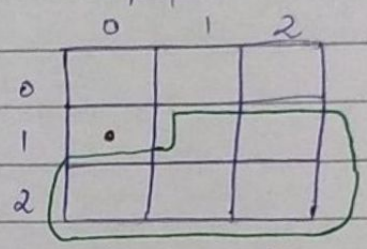
Extra Rec. call



Code :-

```
p s v QueenCombinationBoxRepeat2D (boolean[][] board, int row, int col,
                                   int qpsf, int tq, String ans) {

    // positive base case
    if (qpsf == tq) {
        sysO(ans);
        return;
    }

    // manually var change
    if (col == board[0].length) {
        row++;
        col = 0;
    }

    // negative bc
    if (row == board.length) {
        return;
    }

    // place
    board[row][col] = true;
    QueenCombinationBoxRepeat2D(board, row, col+1, qpsf+1, tq,
                                ans + "{" + row + "-" + col + "}");
    board[row][col] = false;   // undo (backtrack)

    // not place
    QueenCombinationBoxRepeat2D(board, row, col+1, qpsf, tq, ans);
}
```

Imp point



b[1][0] = t

(1,0) Pe queen agar baithe toh iska matlab yeh hai ki iske aage wale rows pe ya aage wale columns queen place nhi hui hogi

Approach - 2

Extra Recursive call

(0,2)

call(1,0)

(1,0)

b[1][0] = t

call(1,1)

return

mandatory

// P // NP ka code na chale

// extra rec call

if(col == board[0].length){

QueenCombinationBoxRespect2D(board, row+1, ~~col~~ 0, qpsf, tq, ans);

return;

}

Very imp. Why?

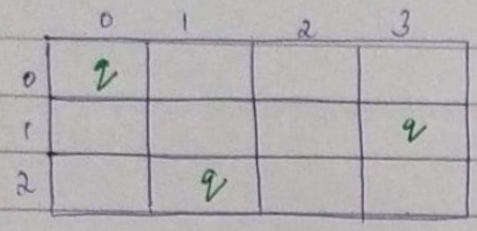
→ agar (0,2) pe ham aa gye hain toh hame // Place & // Not place ki call nhi chalani hai, kyunki (0,2) is out of boundary

Eg. 3×4 board

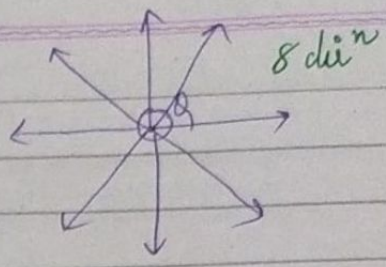
12 cells, 3 queens

$$12C_3 = \frac{\overset{2}{\cancel{12}} \times 11 \times 10}{3 \times \cancel{2}} = 220$$

Queen Combination 2D Kill



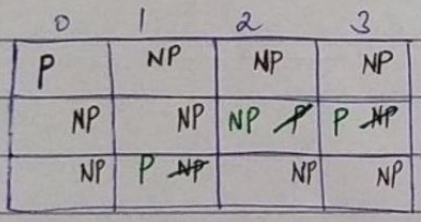
← chessboard
3×4
3 Queens



Queen moves

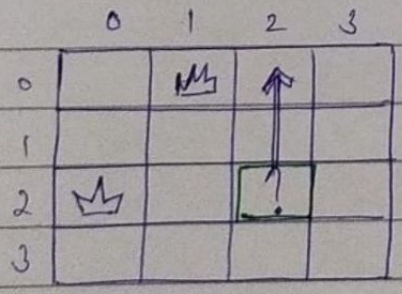
Constraint :- No 2 queens should kill each other

* (00)(01)(02) → Invalid answer
* (00)(01)(03) → Invalid
* (00)(13)(21) → Valid combination



1. Iteration : Black colour (Third queen ko baithna hi nhi paaye)
2. Backtrack karna padega (23) → (22) → (21) → (20) → (13) → (12)
3. After Backtracking, (12) me NP

3. Iteration pe hum tino queens ko place kar paaye (00) (13) (21)



r, c

① ↑ vertically upward check c = col same
r = row − 1

(0,2)
↑
(1,2)

* variables r, c rakhna bohot zaroori hai, Otherwise original variables ka track rakhna dikkat dega, we need original info as it is. Also, we need to reinitialize r & c acc. to every dir^n

Code :-

```
p s boolean isItSafeToPlaceTheQueen(boolean[][] board, int row, int col){

// vertically upward
int r = row-1;
int c = col;
while(r>=0){
    if(board[r][c]){
        return false;
    }
    r--;
}

// horizontally left
r = row;
c = col-1;
while(c>=0){
    Repeat
    c--;
}

// digonally left
r = row-1;
c = col-1;
while(r>=0 && c>=0){
    Repeat
    r--;
    c--;
}
```

Repeat this part

// diagonally right

```
r = row - 1;
c = col + 1;          c < board[0].length
while ( r >= 0 && ~~~~~~ ) {

        Repeat

    r--;
    c++;
}

return true;

}
```

// place only if it is safe

* Queen Combination 2D → copy paste the code with 1 change

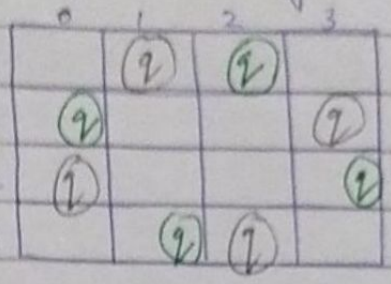
// place only if it safe

```
if ( isItSafeToPlaceTheQueen (board, row, col) ) {

        Same code copy paste

}
```

N Queen (std problem of backtracking)

Eg NxN chessboard
↳ square

N Queens



2 ways (black & green)

5 X 5 Chessboard : 10 ways

* The prev. kill ques that we did was even more generic, any no. of queens, any dimensional board (rectangle/square) was challenged.

* This ques is similar to kill ques that we did previously

* kill (board, row, col, qps, tq, ans)

```
NQueens(board, 0, 0, 0, n, "");
```

↓
square

* But, we can improve the code
Agar suppose (0,0) pe queen ko rakha hai, toh definitely (0,1), (0,2), (0,3) pe queen nhi aa sakti. Toh inke calls do hi mat do to save time

Toh seedhe call (row+1, 0) ispe karlo

row col+1

↑ ↑

row col

If place only if it is safe wrna pass me

Code : github

N Knights

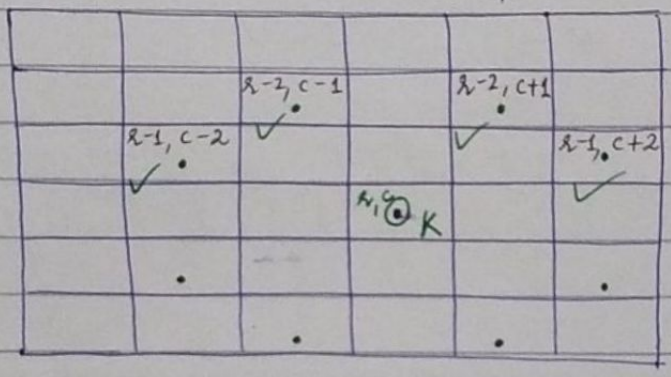
NXN chessboard
N knights
↓
2½ kadam chalte hain

```
rowArr = [-1, -2, -2, -1]
colArr = [ 2,  1, -1, -2]
```

4 posn array

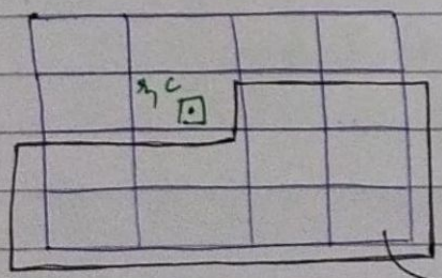
Knight moves

8 places



* If any other knight is at these 8 locations, then K can kill them.

* We had to use while loop in case of NQueens problem when we were writing isItSafe function, kyunki Queen kitne bhi kadam chal sakti hai, but here in knights question, there are only 8 places of danger (kyunki knights ki movement fixed hai, 2½ kadam)

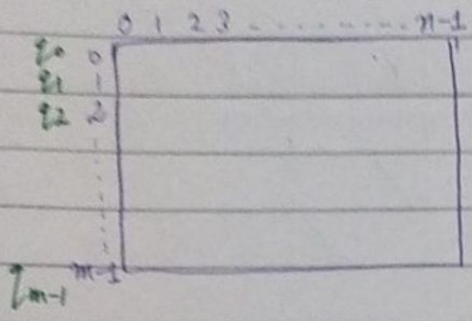


→ Inn places pe abhi tak knight hoga hi nhi (we've discussed this earlier)

Code : github

* Toh basically k ko danger sirf (4) posn hai (as ticked above).
* Neeche wale 4 knight se koi danger nhi hoga.

NQueen 2



m × n CB

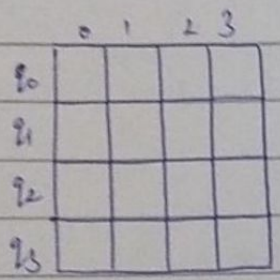
$q_0 q_1 q_2 \dots\dots q_{m-1}$

↓ ↓ ↓ ↓

n n n n

n^m ways

1 row 1 queen



q_3 = 4 ways.

$q_2 q_3$: $4 + 4 + 4 + 4 = 4 \cdot 4 = 4^2$ ways

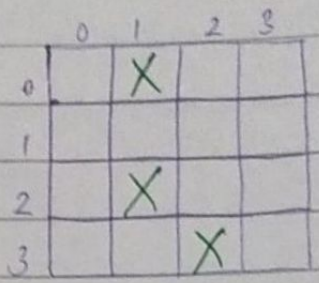
$q_1 q_2 q_3 = 4^2 + 4^2 + 4^2 + 4^2 = 4^2 \cdot 4 = 4^3$ ways

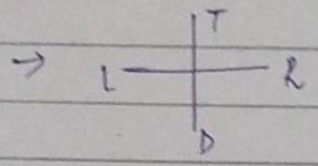
$q_0 q_1 q_2 q_3 = 4^3 + 4^3 + 4^3 + 4^3 = 4^3 \cdot 4 = \boxed{4^4 \text{ ways.}}$

(Har queen ko ek ek row dedicate kedi)

Blocked Maze

→ Maze
→ Blocked : dead end
→ (0,0) → (3,3)





→ ways?

DRRDRD
DRRRDD
DRRTRDDD

3 Ans//

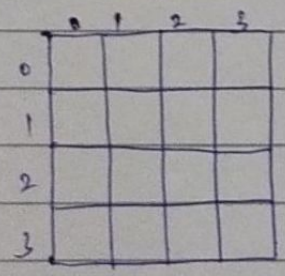
* int[][] maze
blocked = 1

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

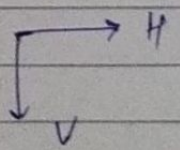
Diff b/w Maze path ques we did & blocked maze :—

Maze Path

1)



2)

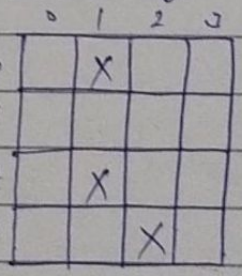


2 options

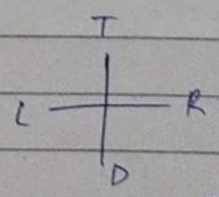
3) +ve moves (kaunsa maze ki dir me hi jaayenge, can't come back)

Blocked Maze

1)



2)

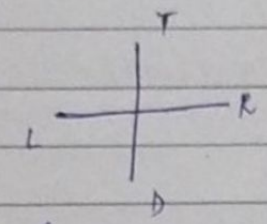


4 options

3) +ve, -ve moves

BP (0,0)

4 options



4 smaller problems

2D array

0	1	0	0
0	0	0	0
0	1	0	0
0	0	1	0

T : (-1, 0)	4SP	row-1, col
D : (1, 0)		row+1, col
L : (0, -1)		row, col-1
R : (0, 1)		row, col+1

Discussion & diff. in BC of two ques

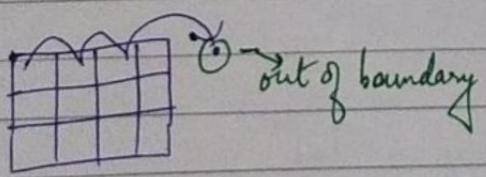
Maze Path

1) +ve base case

row == er && col == ec

2) -ve base case

row > er || col > ec



Yahan pe sirf 2 cond^n lagana sufficient tha

(2 type ke moves)

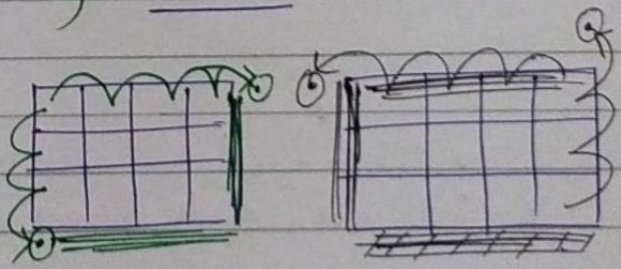
Blocked Maze path

1) +ve bc:

row == m.length - 1 &&

col == m[0].length - 1

2) -ve bc:



row == m.l ||

col == m[0].l

col == -1

row == -1

Yahan pe 4 cond^n lagana zaroori h

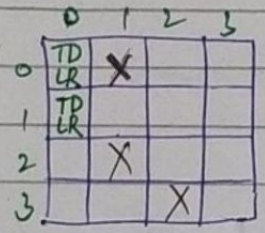
(4 type ke moves)

① → boundary reach : invalid spot
② → blocked cell : invalid spot (maze[row][col] == 1)

Imp. point

* Acha agar yahan tak ka code likh dun aur run karun, toh stack overflow aayega, kyunnn?

* Kyunki hamne ek cheez ka dhyaan toh diya hi nhi — KYA?

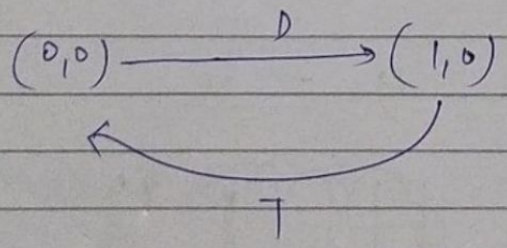


⇒ (0,0) jab Down call karega we'll reach (1,0)

⇒ (1,0) jab Top call karega we'll reach (0,0)

⇒ And the cycle continues....

WHICH RESULTED IN STACK OVERFLOW



infinite rec call.

Yeh kete kete stack memory fill ho jaayegi & LOL

* Toh mark karke aao jahan & jis bhi path se aa rhe ho. Taaki hum wapas us path pe na jaao jahan se aaye the.

* Toh ek 2D boolean array banayenge, visited ke naam se jo ye track rakhegi ki hamne konse cell ko visit kiya hai

(3) → Already visited cell : invalid spot

↳ iske liye ye base case me add krna padega

(visited[row][col])

Code :-

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

    // positive bc
    if(row == maze.length-1 && col == maze[0].length-1){
        syso(ans);
        return;
    }

    // negative bc
    if(row == -1 || row == maze.length || col == -1 || col == mz[0].length ||
       maze[row][col] == 1 || visited[row][col]) {
        return;      // Same return kisi yahan se, can't proceed further (matlab 1,4,D,R calls nhi denge age ke)
    }

    // put a mark
    visited[row][col] = true;
```

```
// T
BlockedMaze(maze, row-1, col, ans + "T", visited);

// D
BlockedMaze(maze, row+1, col, ans + "D", visited);

// L
BlockedMaze(maze, row, col-1, ans + "L" + visited);

// R
BlockedMaze(maze, row, col+1, ans + "R", visited);

}
```