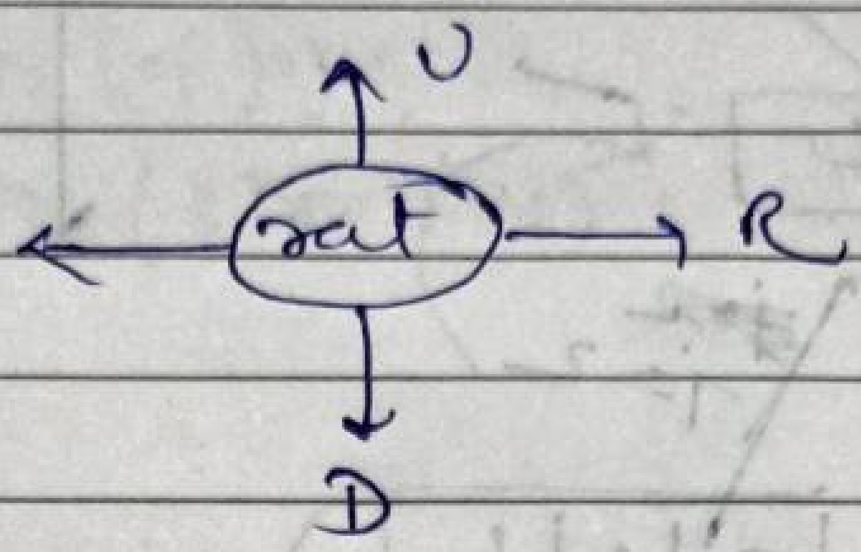
DnC Level-3

Q. Rat In a maze.

0 → Rasta blocked hai
1 → Rasta open hai

src rat	0	0	0
1	1	0	0
1	1	0	0
0	~~1~~0	1	dest

Movement



Q find all solutions to reach destination

DDRDRR, DRDDRR

⇒

src	0	1	0
1	1	1	1
0	1	1	0
0	1	1	0
0	0	1	dest

Find all solutions?

DRDDRDR
DRRDDDDR
DRDRDDR
DRRDLDRDR

} These three directions

	0	1	2	3
0	src	0	0	0
1	1	1	0	0
2	0	1	1	0
3	0	1	1	dest

(0,0) → ~~D~~|L|R|U
↓
(1,0) → ~~D~~|~~L~~|~~R~~|U
↓
(1,1) → ~~D~~|L|R|U
↓
(2,1) → ~~D~~|L|R|U
↓
(3,1) → ~~D~~|L|R|U
↳ infinite Loop
↓
(3,2) → Dest.

To solve infinite loop problem, we'll use 2-D array and mark block as visited once it is visited.

	0	1	2	3
0	1 start	0	0	0
1	1	1	1	0
2	0	1	1	0
3	0	1	1	1 dest

visited

	0	1	2	3
0	1	0	0	0
1	1 $\cancel{0}$	$\cancel{0}$ 1	0	0
2	0	$\cancel{0}$ 1	0	0
3	0	$\cancel{0}$ 1	$\cancel{0}$ 1	0

(0,0) ⟶ $\boxed{D}$ | L | R | U

↓ visited [1] [0] = true.

(1,0) $\cancel{D}$ | $\cancel{L}$ | $\boxed{R}$ | U

↓ visited [1] [1] = true

(1,1) ~~D | L | R | U~~ $\boxed{D}$ | L | R | U

↓ visited [2] [1] = true

(2,1) $\boxed{D}$ | L | R | U

↓ visited [3] [1] = true

(3,1) $\cancel{D}$ | $\cancel{L}$ | $\boxed{R}$ | U

↓ visited [3] [2] = true.

(3,2) $\cancel{D}$ | $\cancel{L}$ | $\boxed{R}$ | U

↓

(3,3) ⟶ Destination.

Safe →

1) index must be inside array

2) index me 1 hona chahiye

3) must be unvisited.

In general

(0,0) ↗ call

↓

(1,1) ⟶ D | L | R | U ⟶ ye sb check hogi

	j-1	j	j+1
i-1	(i-1, j-1)	(i-1, j)	(i-1, j+1)
i	(i, j-1)	(i, j)	(i, j+1)
i+1	(i+1, j-1)	(i+1, j)	(i+1, j+1)

Code

```
int main() {
    int maze[3][3] = {~~{1,1,0,0}, {1,1,1,0}~~ {{1,0,0},
                                               {1,1,0},
if(maze[0][0]==0) {cout<<"No ~~path~~ path"<<endl; {1, 1,1}};
                     return 0; }
    int row = 3;
    int col = 3;

  vector<vector<bool>> visited(row, vector<bool>
  visited[0][0] = true;             (col, false));
  vector<string> path;
  string output = "";
  solveMaze(~~arr~~ maze, row, 0, 0, visited, path, output);
  ~~return 0;~~ cout<< "Printing results" << endl;
}                for(auto i: path){    cout<<endl;
                 cout<<i<<" "; }       int col.
void solveMaze(int arr[3][3], int row, int srcx
        , int ~~r~~ srcy, vector<vector<bool>>&
            visited, vector<string& path, string
            output) {
    if (srcx == row-1 && srcy == col-1){
        path.push_back(output);
        return;
    }
```

```
//Ek case solve karo.
//Down → i+1, j

if(issafe(i+1, j, row, col, arr, visited)) {
    visted[i+1][j]= true;
   solveMaze (arr, row, col, i+1, j, visited,
                  output + 'D');
    visited[i+1][j] = false;
}

//~~Right~~ Left → i, j-1
if (is safe (i, j-1, row, col, arr, visited)) {
      visited[i][j-1] = true;
 solveMaze (arr, row, col, i, j-1, visited, output +
                                                    'L';
     visited[ i ][j-1] = false;
}
 // Right → i, j+1
if (issafe ( i, j+1, row, col, arr, visited)) {
       visited [i][j+1] = true;
solvemaze (arr, row, col, i, j+1, visited,
             output + 'R';
    visited [i][j+1] = false;
}

// Up → [i-1][j]
 if (is safe( i-1, j, row, col, arr, visited)) {
      visited[i-1][j]= true;
solvemaze ( arr, row, col, i-1, j, visited,
               output + 'U';
     visited[i-1][j] = false.
 }
```

Scanned by TapScanner

PAGE No
DATE

```
bool isSafe(int ~~x~~i, int ~~y~~j, int row, int col
        int arr[][3], vector<vector<bool>> &
        visited) {
    if(((i>=0) && j<row) && (~~y~~ i>=0 && j<col)) &&
        (arr[i][j]==1) &&
        (visited[i][j]==false)) {
        return true;
    }
    else {
        return false;
    }
}
```