22/03/2023

Rat in a maze

This is a very famous question of the backtracking.

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

} Maze

Rat needs to go from src to dest.
0 → Path is blacked
1 → Path is open
Rat can move right, left, down & up only. Here we will be finding all the solutions.

1st solution DDRDRR
2nd solution DRDDRR
We won't be finding useless solutions i.e. going forward & then going back again, this is an example of useless solution.

Problem which we need to handle

(0,0) → Down / Left / Right / Up
↑ Selected

(1,0) → Down / Left / Right / Up
× × ↳ Selected

(1,1) → Down / Left / Right / Up
↑ Selected

(2,1) → Down / Left / Right / Up
↳ Selected ↑ Selected

(3,1) → Down / Left / Right / Up
× ×

(3,2) → Down / Left / Right / up
×
↳ Selected (Now we again reach to (3,1)) → ∞ loop

To fix the issue of infinite loop, we will be creating a visited 2D array & we will explore the path only when visited is marked as false. It is safe to move to next position only when the following 3 conditions are satisfied.

1) index is a valid one i.e inside the array.
2) next position has 1 i.e path is open.
3) visited array should have false value i.e. not visited.

The modification that we will doing in the approach to solve the problem is that once we reach the next position, mark it visited by placing true in the visited array.

Base case → The base case is when we reach the destination i.e. we have found one solution & then return to explore other solutions & while returning back we need to make the corresponding position in the visited array as false & this is backtracking line.

Note → (i, j)

D → (i+1, j)
L → (i, j-1)
R → (i, j+1)
U → (i-1, j)

	j-1	j	j+1
i-1		↑U	
i	←L	(i,j)	→R
i+1		↓D	

Code

```
bool isSafe (int x, int y , int row, int
col, int arr [][3], vector <vector <bool>>
& visited) {
                                valid index (1st cond)
if ( (x >=0 && x < row)  && (y >=0 && y <col)
      && (arr [x][y] == =1) && (visited [x][y]
      == = false ))   ↳ 2nd cond      ↓ 3rd cond
{
  return true;
}
else {
    return false;
}
}

void solveMaze (int arr [3][3], int row,
int col , int i, int j , vector <vector <bool>>
&visited , vector <string>&path , string output)
{
    // Base case
    if (i == row-1 && j == col-1){
        // Destination reached
        path.push-back (output);
        return;
    }
    // Solve one case
    // Down
    if (isSafe (i+1, j, row , col, arr , visited)){
Mark visited ← visited [i+1][j] = true;
  true  {solveMaze (arr, row, col , i+1, j , visited,
Recursive call      path , output + 'D');
```

```
        // Backtracking line
        visited[i+1][j] = false; → Recreating original state
}
// Left
if (isSafe(i, j-1, row, col, arr, visited)) {
        visited[i][j-1] = true; → mark visited true
Recursive call { solveMaze(arr, row, col, i, j-1, visited,
                 path, output + 'L');
        //Backtracking
        visited[i][j-1] = false; → Recreating original state
}
// Right
if (isSafe(i, j+1, row, col, arr, visited)) {
        visited[i][j+1] = true; → mark visited true
Recursive call { solveMaze(arr, row, col, i, j+1,
                 visited, path, output + 'R');
        //Backtracking
        visited[i][j+1] = false; → Recreating original state
}
// Up
if (isSafe(i-1, j, row, col, arr, visited)) {
        visited[i-1][j] = true; → mark visited true
Recursive call { solveMaze(arr, row, col, i-1, j,
                 visited, path, output + 'U');
        // Backtracking
        visited[i-1][j] = false; → Recreating original state
    }
}
```

Note → Check in main

(i) maze[0][0] == 0 i.e src can't be reached & hence no path exist.

vector of strings

(ii) Print path in main.
(iii) If path has nothing, then no path exists.

Time Complexity

For each & every case, there can be at max 4 function calls just like in fibonacci, at max 2 function calls were there. Hence this will have exponential time complexity.

$TC = O(4^n)$