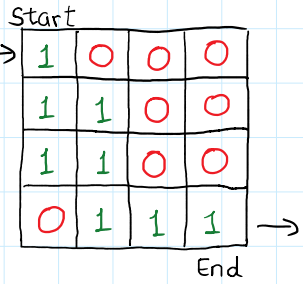
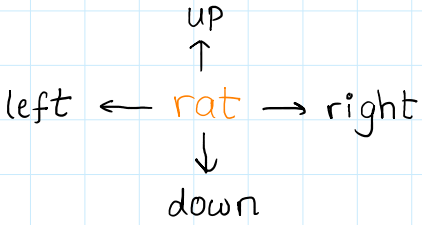
PROBLEM LINK : Click Here

Given :

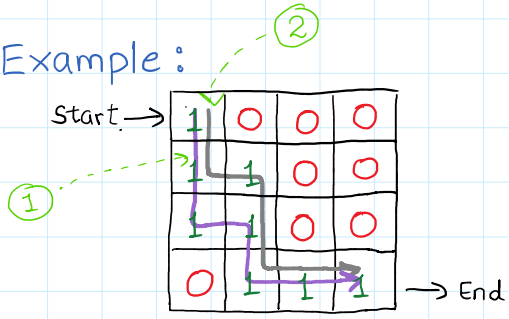


where : 1 - we can come to this cell
0 - we cannot come to this cell.

Aim is to output all possible ways to reach the End (n-1, n-1) from the Start (0, 0). The rat can move in all the 4 directions up, down, left, right.



Example :



1. DDRDRR
2. DRDDRR

APPROACH :

At any point, we have 4 choices – up, down, left or right.

For each position, check if you can go U/D/L/R keeping in mind the following constraints :

① Don't go out of bounds. ($0 \le i < n$, $0 \le j < n$)

② Don't go to an already visited position.

③ Don't go to a position marked 0.

③ Don't go to a position marked 0.

For point ②, maintain a 2-D array 'visited' which keeps track of all the visited positions. (nxn size)

Once you reach (n-1, n-1), you can print the string showing the path you took and then make all cells of the 'visited' matrix as False.

Thus, when you move, you will make visited[i][j] (if you moved to m[i][j]) and you will append your string with D, U, L or R according to your movement.

CODE: To check if a position is legit, we discussed some conditions above.

```
bool isSafe(int x, int y, int n, vector<vector<int>> visited, vector<vector<int>> &m ) {

    if( (x>=0 && x<n ) && (y>=0 && y<n) && visited[x][y] == 0 && m[x][y] == 1) {
        return true;
    }
    else {
        return false;
    }
}
```

In our main 'solve' function, we will first check if we have reached the end of the maze.

```
void solve(vector<vector<int>> &m, int n,vector<string>& ans, int x,
           int y, vector<vector<int>> visited , string path) {

    //you have reached x,y here

    //base case
    if( x == n-1 && y == n-1){
        ans.push_back(path);
        return;
    }
```

For the position (x, y), mark visited[i][j] = 1 (or true)

```
visited[x][y] = 1;
```

Now, for each direction (U, D, L, R) make newx & newy and make a move if going in that direction is safe.

if path is safe, then append the move in the path string.

```
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('D');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}
```

Call for the new position

After the recursion call returns, remove this move to make way for the next option. (U/D/L/R)

This gets called 4 times:

1. newx = x, newy = y-1
2. newx = x-1, newy = y
3. newx = x, newy = y+1
4. newx = x+1, newy = y.

```
int newx = x+1;
int newy = y;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('D');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}

//left
newx = x;
newy = y-1;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('L');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}

//Right
newx = x;
newy = y+1;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('R');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}

//UP
newx = x-1;
newy = y;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('U');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}
```

After this, we have covered all possible paths from (x,y) to (n-1, n-1). So, we will now backtrack and make visited[x][y] = 0 (or False)

```
visited[x][y] = 0;
```