DAY 15

1)

def solve\_n\_queens(n):

def is\_safe(row, col, cols\_used, diag\_used, anti\_diag\_used):

return (col not in cols\_used

and (row - col) not in diag\_used

and (row + col) not in anti\_diag\_used)

def backtrack(row, cols\_used, diag\_used, anti\_diag\_used, board, solutions):

if row == n:

solutions.append([''.join(r) for r in board])

return

for col in range(n):

if is\_safe(row, col, cols\_used, diag\_used, anti\_diag\_used):

board[row][col] = 'Q'

cols\_used.add(col)

diag\_used.add(row - col)

anti\_diag\_used.add(row + col)

backtrack(row + 1, cols\_used, diag\_used, anti\_diag\_used, board, solutions)

board[row][col] = '.'

cols\_used.remove(col)

diag\_used.remove(row - col)

anti\_diag\_used.remove(row + col)

solutions = []

board = [['.'] \* n for \_ in range(n)]

backtrack(0, set(), set(), set(), board, solutions)

return solutions;

2)

def find\_maze\_paths(maze, n):

def is\_safe(x, y, visited):

return (0 <= x < n and 0 <= y < n

and maze[x][y] == 1

and not visited[x][y])

def backtrack(x, y, path, visited, solutions):

if x == n - 1 and y == n - 1:

solutions.append(path)

return

visited[x][y] = True

for direction, dx, dy in directions:

nx, ny = x + dx, y + dy

if is\_safe(nx, ny, visited):

backtrack(nx, ny, path + direction, visited, solutions)

visited[x][y] = False

solutions = []

directions = [('D', 1, 0), ('L', 0, -1), ('R', 0, 1), ('U', -1, 0)]

visited = [[False] \* n for \_ in range(n)]

if maze[0][0] == 1:

backtrack(0, 0, "", visited, solutions)

return solutions

3)

def letter\_combinations(digits):

if not digits:

return []

digit\_to\_letters = {

'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',

'6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'

}

def backtrack(index, path, solutions):

if index == len(digits):

solutions.append("".join(path))

return

for letter in digit\_to\_letters[digits[index]]:

path.append(letter)

backtrack(index + 1, path, solutions)

path.pop()

solutions = []

backtrack(0, [], solutions)

return solutions