RAT-MAZE problem using backtracking

In [22]:
```
directions = 'DLRU'

dr = [1, 0, 0, -1]
dl = [0, -1, 1, 0]

def isValid(r, c, maze):
    if r < 0 or r >= len(maze):
        return False

    if c < 0 or c >= len(maze[0]):
        return False

    if maze[r][c] != 0:
        return False

    return True

def findPath(r, c, maze, path):
    if r == len(maze) - 1 and c == len(maze[0]) - 1:
        return True

    valid_cell = isValid(r, c, maze)

    if valid_cell:
        maze[r][c] = 0

        for i in range(len(directions)):
            next_row = r + dr[i]
            next_col = c + dl[i]
            if findPath(next_row, next_col, maze, path + directions[i]):
                return True
        maze[r][c] = 1

    return False


n = 4
maze = [
    [1, 0, 0, 0],
    [1, 1, 0, 1],
    [1, 1, 0, 0],
    [0, 1, 1, 1]
]

result = []

current_path = ''
findPath(0, 0, maze, current_path)

if len(current_path) == 0:
    print('No path found')
else:
    print(current_path)
```

```
No path found
```

N QUEENS PROBLEM

In [9]:
```
n = 5

def isSafe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False

    i = row
    j = col
    while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return False
        i -= 1
        j -= 1

    i = row
    j = col
    while i < n and j >= 0:
        if board[i][j] == 1:
            return False
        i += 1
        j -= 1

    return True


def solve(board, col):
    if col >= n:
        return True

    for i in range(n):
        if isSafe(board, i, col):
            board[i][col] = 1

            if solve(board, col + 1) == True:
                return True

            board[i][col] = 0

    return False


def final_solve():
    board = [[0 for i in range(n)] for j in range(n)]

    if solve(board, 0) == False:
        print("No solution exists")
    else:
        print(board)

final_solve()
```

```
[[1, 0, 0, 0, 0], [0, 0, 0, 1, 0], [0, 1, 0, 0, 0], [0, 0, 0, 0, 1], [0, 0, 1, 0, 0]]
```

SUBSET SUM

In [21]:
```
def subsetsum(nums,target):
    def backtrack(start,path,curr_sum):
        if curr_sum==target:
            print(path)
            result.append(path)
            return
        if curr_sum>target:
            return
        for i in range(start,len(nums)):
            path.append(nums[i])
            backtrack(i+1,path,curr_sum+nums[i])
            path.pop()

    result = []
    backtrack(0,[],0)


nums = [1, 2, 3, 4, 5]
target = 7
subsetsum(nums, target)
```

```
[1, 2, 4]
[2, 5]
[3, 4]
```

GRAPH COLORING PROBLEM

In [24]:
```
def isSafe(graph, node, color, color_assignment):
    for neighbor in graph[node]:
        if color_assignment[neighbor] == color:
            return False
    return True

def graphColoring(graph, colors, node, color_assignment):
    if node == len(graph):
        return True

    for color in colors:
        if isSafe(graph, node, color, color_assignment):
            color_assignment[node] = color

            if graphColoring(graph, colors, node + 1, color_assignment):
                return True

            color_assignment[node] = None

    return False


graph = {
    0: [1, 2],
    1: [0, 2, 3],
    2: [0, 1, 3],
    3: [1, 2]
}

colors = ['Red', 'Green', 'Blue']
color_assignment = [None] * len(graph)

if graphColoring(graph, colors, 0, color_assignment):
    print("Graph can be colored")
    print("Color assignment:", color_assignment)
else:
    print("Graph cannot be colored")
```

```
Graph can be colored
Color assignment: ['Red', 'Green', 'Blue', 'Red']
```

HAMILTONIAN CYCLE

In [25]:
```
def isSafe(v, graph, path, pos):
    if graph[path[pos-1]][v] == 0:
        return False

    if v in path:
        return False

    return True

def hamiltonianCycleUtil(graph, path, pos):
    if pos == len(graph):
        if graph[path[pos-1]][path[0]] == 1:
            return True
        else:
            return False

    for v in range(1, len(graph)):
        if isSafe(v, graph, path, pos):
            path[pos] = v

            if hamiltonianCycleUtil(graph, path, pos+1):
                return True

            path[pos] = -1

    return False

def hamiltonianCycle(graph):
    path = [-1] * len(graph)
    path[0] = 0

    if hamiltonianCycleUtil(graph, path, 1):
        print("Hamiltonian cycle exists")
        print("Path:", path)
    else:
        print("Hamiltonian cycle does not exist")

graph = [
    [0, 1, 0, 1, 0],
    [1, 0, 1, 1, 1],
    [0, 1, 0, 0, 1],
    [1, 1, 0, 0, 1],
    [0, 1, 1, 1, 0]
]

hamiltonianCycle(graph)
```

```
Hamiltonian cycle exists
Path: [0, 1, 2, 4, 3]
```