1.N QUEEN PROBLEM.

PROGRAM: def is\_safe(board, row, col, N):

for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_n\_queens\_util(board, col, N):

if col >= N:

return True

for i in range(N):

if is\_safe(board, i, col, N):

board[i][col] = 1

if solve\_n\_queens\_util(board, col + 1, N) == True:

return True

board[i][col] = 0

return False

def solve\_n\_queens(N):

board = [[0 for \_ in range(N)] for \_ in range(N)]

if solve\_n\_queens\_util(board, 0, N) == False:

return False

return board

def print\_solution(board):

for row in board:

print(row)

# Example usage

N = 4

solution = solve\_n\_queens(N)

if solution:

print\_solution(solution)

else:

print("No solution exists for N = ", N)

2.SUBSET SUM.

PROGRAM: def isSubsetSum(arr, n, sum):

if sum == 0:

return True

if n == 0 and sum != 0:

return False

if arr[n-1] > sum:

return isSubsetSum(arr, n-1, sum)

return isSubsetSum(arr, n-1, sum) or isSubsetSum(arr, n-1, sum-arr[n-1])

arr = [3, 34, 4, 12, 5, 2]

sum = 9

n = len(arr)

if isSubsetSum(arr, n, sum) == True:

print("Found a subset with the given sum")

else:

print("No subset with the given sum")

3.GRAPH COLORING.

PROGRAM: import networkx as nx

import matplotlib.pyplot as plt

G = nx.Graph()

G.add\_edges\_from([(1, 2), (1, 3), (2, 3), (3, 4), (4, 5), (4, 6), (5, 6)])

colors = nx.greedy\_color(G, strategy='largest\_first')

pos = nx.spring\_layout(G)

nx.draw(G, pos, with\_labels=True, node\_color=[colors[node] for node in G.nodes], cmap=plt.get\_cmap('viridis'))

plt.show()

4.HAMILTONIAM CIRCUIT PROBLEM.

PROGRAM: import networkx as nx

from itertools import permutations

# Create a graph

G = nx.Graph()

G.add\_nodes\_from([1, 2, 3])

G.add\_edges\_from([(1, 2), (2, 3), (3, 1)])

# Find all possible Hamiltonian cycles

hamiltonian\_cycles = [list(perm) for perm in permutations(G.nodes)]

for cycle in hamiltonian\_cycles:

if (cycle[0], cycle[-1]) in G.edges:

cycle\_edges = list(zip(cycle, cycle[1:] + [cycle[0]]))

if set(cycle\_edges).issubset(G.edges):

print("Hamiltonian Cycle found:", cycle)

break

5.PERMUTATION N COMPUTATION.

PROGRAM: import itertools

n = 4

permutations = list(itertools.permutations(range(1, n+1)))

print(permutations)

6.SUDOKU SOLVER.

PROGRAM: def is\_valid\_move(board, row, col, num):

for i in range(9):

if board[row][i] == num or board[i][col] == num:

return False

start\_row, start\_col = 3 \* (row // 3), 3 \* (col // 3)

for i in range(3):

for j in range(3):

if board[i + start\_row][j + start\_col] == num:

return False

return True

def solve\_sudoku(board):

empty = find\_empty\_location(board)

if not empty:

return True

row, col = empty

for num in range(1, 10):

if is\_valid\_move(board, row, col, num):

board[row][col] = num

if solve\_sudoku(board):

return True

board[row][col] = 0

return False

def find\_empty\_location(board):

for i in range(9):

for j in range(9):

if board[i][j] == 0:

return (i, j)

return None

# Example Sudoku board

board = [

[5, 3, 0, 0, 7, 0, 0, 0, 0],

[6, 0, 0, 1, 9, 5, 0, 0, 0],

[0, 9, 8, 0, 0, 0, 0, 6, 0],

[8, 0, 0, 0, 6, 0, 0, 0, 3],

[4, 0, 0, 8, 0, 3, 0, 0, 1],

[7, 0, 0, 0, 2, 0, 0, 0, 6],

[0, 6, 0, 0, 0, 0, 2, 8, 0],

[0, 0, 0, 4, 1, 9, 0, 0, 5],

[0, 0, 0, 0, 8, 0, 0, 7, 9]

]

if solve\_sudoku(board):

for row in board:

print(row)

else:

print("No solution exists.")