**GROUP A**

**Title:** Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph colouring problem.

1. **N-queen Problem**

**Program:**

global N

N = 4

def printSolution(board):

for i in range(N):

for j in range(N):

print(board[i][j],end = " ")

print()

def isSafe(board, row, col):

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] == 2:

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 solveNQUtil(board,col):

if col >= N:

return True

for i in range(N):

if isSafe(board,i,col):

board[i][col] = 1

if solveNQUtil(board,col + 1) == True:

return True

board[i][col] = 0

return False

def solveNQ():

board = [ [0,0,0,0],

[0,0,0,0],

[0,0,0,0],

[0,0,0,0]

]

if solveNQUtil(board,1) == False:

print("Solution does not exist")

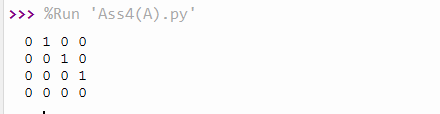
return False

printSolution(board)

return True

solveNQ()

**Output:**



1. **Graph Colouring Problem**

**Program:**

class Graph:

def \_\_init\_\_(self,edges,n):

self.adjList = [[] for \_ in range(n)]

for (src,dest) in edges:

self.adjList[src].append(dest)

self.adjList[dest].append(src)

def colorGraph(graph,n):

result = {}

for u in range(n):

assigned = set([result.get(i) for i in graph.adjList[u] if i in result])

color = 1

for c in assigned:

if color != c:

break

color = color + 1

result[u] = color

for v in range(n):

print(f'Color assigned to vertex {v} is {colors[result[v]]}')

if \_\_name\_\_ == '\_\_main\_\_':

colors = [",'BLUE','GREEN','RED','YELLOW','ORANGE','PINK','BLACK','BROWN','WHITE','PURPLE','VIOLET'"]

edges = [(0,1),(0,4),(0,5),(4,5),(1,4),(1,3),(2,3),(2,4)]

n = 8

graph = Graph(edges,n)

colorGraph(graph,n)

**Output:**

