AI-4:

class NQueens:

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

self.n = n

self.board = [[0] \* n for \_ in range(n)]

self.solutions = []

def is\_safe(self, row, col):

for i in range(col):

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

return False

if row - i - 1 >= 0 and self.board[row - i - 1][col - i - 1] == 1:

return False

if row + i + 1 < self.n and self.board[row + i + 1][col - i - 1] == 1:

return False

return True

def solve\_backtracking(self, col):

if col >= self.n:

self.solutions.append([row[:] for row in self.board])

return True

for i in range(self.n):

if self.is\_safe(i, col):

self.board[i][col] = 1

self.solve\_backtracking(col + 1)

self.board[i][col] = 0

return False

def solve\_branch\_and\_bound(self, col):

if col >= self.n:

self.solutions.append([row[:] for row in self.board])

return True

for i in range(self.n):

if self.is\_safe(i, col):

self.board[i][col] = 1

self.solve\_branch\_and\_bound(col + 1)

self.board[i][col] = 0

return False

def print\_solutions(self):

for solution in self.solutions:

print("Solution:")

for row in solution:

print(row)

print()

# Example usage:

n\_queens = NQueens(4)

print("Backtracking:")

n\_queens.solve\_backtracking(0)

n\_queens.print\_solutions()

n\_queens = NQueens(4)

print("Branch and Bound:")

n\_queens.solve\_branch\_and\_bound(0)

n\_queens.print\_solutions()

O/ P

Backtracking:

Solution:

[0, 1, 0, 0]

[0, 0, 0, 1]

[1, 0, 0, 0]

[0, 0, 1, 0]

Solution:

[0, 0, 1, 0]

[1, 0, 0, 0]

[0, 0, 0, 1]

[0, 1, 0, 0]

Branch and Bound:

Solution:

[0, 1, 0, 0]

[0, 0, 0, 1]

[1, 0, 0, 0]

[0, 0, 1, 0]

Solution:

[0, 0, 1, 0]

[1, 0, 0, 0]

[0, 0, 0, 1]

[0, 1, 0, 0]