**Experiment No: 7**

***Code for Hill Climbing algorithm for 4-queen:***

import random

import numpy as np

N\_QUEEN = 4

def in\_conflict(column, row, other\_column, other\_row):

if column == other\_column:

return True # Same column

if row == other\_row:

return True # Same row

if abs(column - other\_column) == abs(row - other\_row):

return True # Diagonal

return False

def in\_conflict\_with\_another\_queen(row, column, board):

for other\_column, other\_row in enumerate(board):

if in\_conflict(column, row, other\_column, other\_row):

if row != other\_row or column != other\_column:

return True

return False

def count\_conflicts(board):

cnt = 0

for queen in range(0, len(board)):

for other\_queen in range(queen + 1, len(board)):

if in\_conflict(queen, board[queen], other\_queen, board[other\_queen]):

cnt += 1

return cnt

def evaluate\_state(board):

return (len(board) - 1) \* len(board) / 2 - count\_conflicts(board)

def print\_board(board):

for row in range(len(board)):

line = ''

for column in range(len(board)):

if board[column] == row:

line += 'Q' if in\_conflict\_with\_another\_queen(row, column, board) else 'q'

else:

line += '\_'

print(line)

print("")

def init\_board(nqueens):

board = []

for column in range(nqueens):

board.append(random.randint(0, nqueens - 1))

print('Initial State:')

print\_board(board)

return board

def Hill\_Climbing(board):

i = 0

optimum = (len(board) - 1) \* len(board) / 2

evaluation = [evaluate\_state(board)]

while evaluate\_state(board) != optimum:

i += 1

max\_score\_of\_each\_column = []

row\_resulting\_in\_max\_score = []

for col in range(len(board)):

col\_scores = []

for row in range(len(board)):

new\_board = board.copy()

new\_board[col] = row

col\_scores.append(evaluate\_state(new\_board))

if max(col\_scores) > evaluate\_state(board):

max\_score\_of\_each\_column.append(max(col\_scores))

row\_resulting\_in\_max\_score.append(np.argmax(col\_scores))

else:

max\_score\_of\_each\_column.append(False)

row\_resulting\_in\_max\_score.append(False)

if max(max\_score\_of\_each\_column):

maximizing\_col = np.argmax(max\_score\_of\_each\_column)

maximizing\_row = row\_resulting\_in\_max\_score[maximizing\_col]

board[maximizing\_col] = maximizing\_row

evaluation.append(evaluate\_state(board))

if evaluate\_state(board) == optimum:

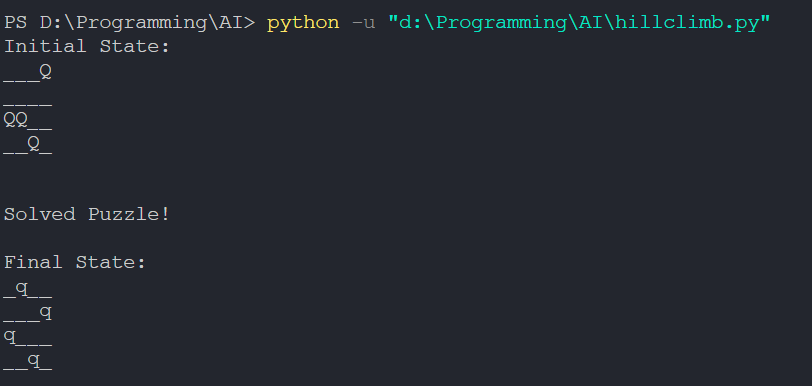
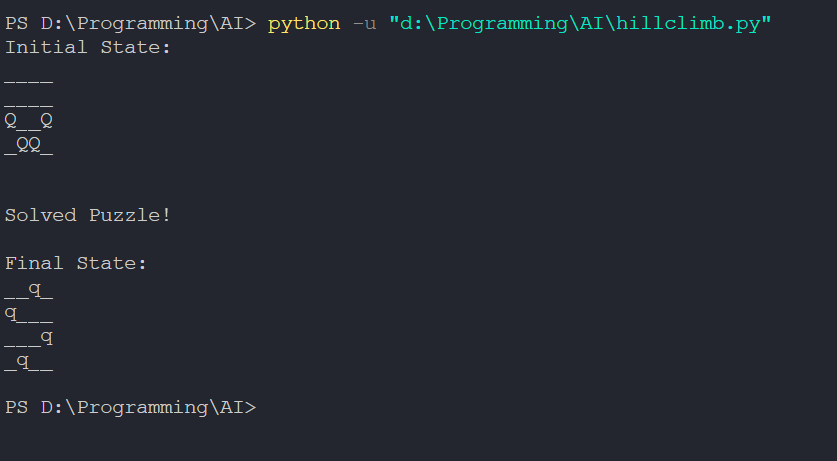
print('\nSolved Puzzle!')

print('\nFinal State:')

print\_board(board)

if \_\_name\_\_ == "\_\_main\_\_":

Hill\_Climbing(init\_board(N\_QUEEN))

***Output:***

**\**

***\*\*\*\*\****