**IMPLEMENTATION OF CONSTRAINT SATISFACTION PROBLEMS**

**AIM**:

Program a solution to N-Queens Problem using Python

**PROCEDURE**:

1. Import itertools.
2. Create a function to take note of empty positions in the state space and update while getting filled.
3. Create a function to properly order the remaining positions so that they are not repeated.
4. Create a recursive generator function to allocate the position of queens in accordance to the constraints.
5. Create a print function to display all possible state spaces which satisfy the aforementioned constraints.
6. Create print statement to display the no. of solutions.
7. Run the program, input the no. of queens.
8. Check the output.

**Constraints:**

The **implicit constraints** are that no two 's can be the same (as **queens** must be on different columns) and no two **queens** can be on the same diagonal. this implies that all solutions are permutations of the **8**-tuple (1,2,…,**8**), and reduces the solution space from tuples to **8**!

**CODE:**

import itertools

def rempos(board, size):

    takenr = frozenset(x for x, y in board)

    freer = (x for x in range(size) if x not in takenr)

    takenc = frozenset(y for x, y in board)

    freec = (y for y in range(size) if y not in takenc)

    freerc = itertools.product(freer, freec)

    taken\_first\_diagonals = frozenset(x + y for x, y in board)

    taken\_second\_diagonals = frozenset(x - y for x, y in board)

    return (

        (x, y) for x, y in freerc

        if x + y not in taken\_first\_diagonals

        and x - y not in taken\_second\_diagonals

    )

def ordrem(board, size):

    if board:

        max\_x = max(x for x, y in board)

    else:

        max\_x = -1

    return (

        (x, y) for x, y in rempos(board, size)

        if x > max\_x

    )

def nqueens(size, board=None):

    board = board or frozenset()

    if len(board) == size:

        yield board

    for position in ordrem(board, size):

        new\_board = board.union((position,))

        yield from nqueens(size, new\_board)

def printb(board):

    size = len(board)

    for row in range(size):

        cells = (

            "Q " if (row, col) in board else "\_ "

            for col in range(size)

        )

        print("".join(cells))

    print("               ")

if \_\_name\_\_ == "\_\_main\_\_":

    size = int(input("Please enter the no. of queens: "))

    print("The size of board will be: ",size)

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("Constraints placed: On the board of size",size,",",size,"queens must be placed in such a way that no 2 queens co-incide horizantally, vertically, or diagonally")

    print("\n\n")

    print("The solutions are as follows: ")

    solution\_count = 0

    for solution in nqueens(size):

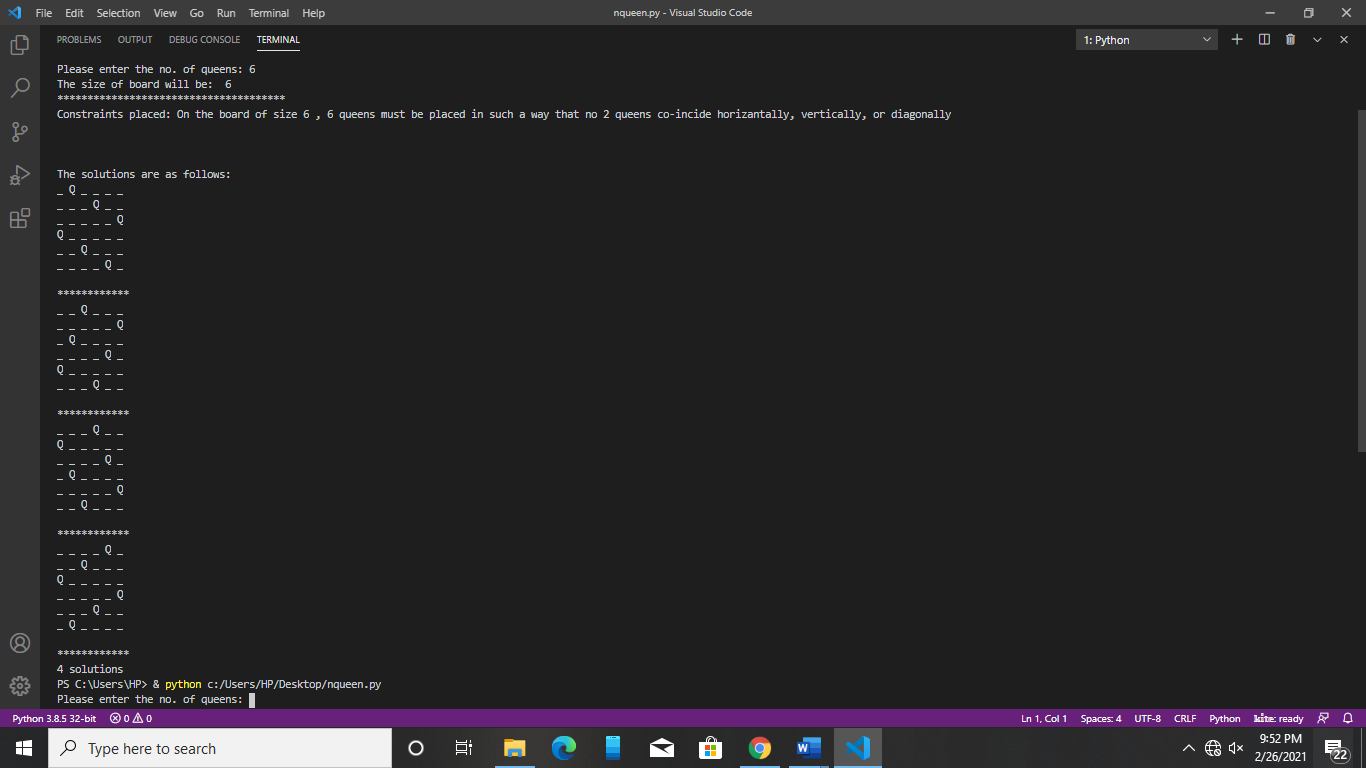
        solution\_count += 1

        printb(solution)

        print("\*\*" \* size)

    print("{count} solutions".format(count=solution\_count))

**OUTPUT:**



**RESULT**

Thus, the implementation of N-Queens problem using Python was performed successfully.