Open | DAA_Program5.py | Save

```python
# Python3 program to solve N Queen
# Problem using backtracking
global N
N = 4

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only left side for
# attacking queens
def isSafe(board, row, col):

    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    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):

    # base case: If all queens are placed
    # then return true
    if col >= N:
        return True

    # Consider this column and try placing
    # this queen in all rows one by one
    for i in range(N):

        if isSafe(board, i, col):

```

Python 2 | Tab Width: 4 | Ln 18, Col 29 | INS

```
40
41      # base case: If all queens are placed
42      # then return true
43      if col >= N:
44          return True
45
46      # Consider this column and try placing
47      # this queen in all rows one by one
48      for i in range(N):
49
50          if isSafe(board, i, col):
51
52              # Place this queen in board[i][col]
53              board[i][col] = 1
54
55              # recur to place rest of the queens
56              if solveNQUtil(board, col + 1) == True:
57                  return True
58
59              # If placing queen in board[i][col
60              # doesn't lead to a solution, then
61              # queen from board[i][col]
62              board[i][col] = 0
63
64      # if the queen can not be placed in any row in
65      # this column col then return false
66      return False
67
68  # This function solves the N Queen problem using
69  # Backtracking. It mainly uses solveNQUtil() to
70  # solve the problem. It returns false if queens
71  # cannot be placed, otherwise return true and
72  # placement of queens in the form of 1s.
73  # note that there may be more than one
74  # solutions, this function prints one of the
75  # feasible solutions.
76  def solveNQ():
77      board = [ [0, 0, 0, 0],
78                [0, 0, 0, 0],
79                [0, 0, 0, 0],
80                [0, 0, 0, 0] ]
81
82      if solveNQUtil(board, 0) == False:
83          print ("Solution does not exist")
84          return False
85
86      printSolution(board)
87      return True
88
89  # Driver Code
90  solveNQ()
```


ubuntu@linux: ~
ubuntu@linux:~$ python3 DAA_Program5.py
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
ubuntu@linux:~$