

Artificial Intelligence

Lab 1

N- Queens Problem:

The N queen's problem can be stated as follows. Consider a $n \times n$ chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

Aim: To solve N- Queens Problem.

Algorithm: Backtracking Algorithm

1) Start in the leftmost column

2) If all queens are placed and **return true**

print the solution.

3) Try all rows in the current column.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing the queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

3) If all rows have been tried and nothing worked, **return false** to trigger backtracking.

Code :

```

global n
global answer

def printSolution(board):
    for r in board:
        print(str(r).replace(',', ' ').replace('\n', ''))
    print()

def isSafe(board, row, column):
    # print(board)
    r = row
    c = column
    # upper diagonal
    while r >= 0 and c >= 0:
        if board[r][c] == 'Q':
            return False
        r -= 1
        c -= 1

    r = row
    c = column

    # left side
    while c >= 0:
        if board[r][c] == 'Q':
            return False
        c -= 1

    r = row
    c = column

    # lower diagonal
    while r < n and c >= 0:
        if board[r][c] == 'Q':
            return False
        r += 1
        c -= 1

    return True

def getAnswer(board, col):
    if col == n:
        # print(n)
        printSolution(board)
        return

    for row in range(0, n):
        if isSafe(board, row, col):
            board[row][col] = 'Q'
            getAnswer(board, col + 1)
            board[row][col] = '.'

def solveNQueens(n):
    board = [['.' * n for _ in range(n)]]
    getAnswer(board, 0)

if __name__ == "__main__":
    n = int(input("Enter number of queens"))
    print("Different possible positions with given " + str(n) + " queens are :")
    count = 0
    solveNQueens(n)

```

N Queens Problem

Output :

$N = 4$

File Edit Find View Go Run Tools Window Support Preview Run

Go to Anything (Ctrl-P)

Dr.Ashish M Namb

RA1911026010032

RA1911026010033

RA1911026010034

RA1911026010035

RA1911026010036

RA1911026010037

RA1911026010038

RA1911026010040

RA1911026010041

RA1911026010042

RA1911026010043

RA1911026010044

RA1911026010045

RA1911026010046

RA1911026010047

README.md

nQueens.py

```
25 if board[r][c] == 'Q':
26     return False
27 c -= 1
28
29 r = row
30 c = column
31
32 # Lower diagonal
33 while r < n and c >= 0:
34     if board[r][c] == 'Q':
35         return False
36     r += 1
37     c -= 1
38
39 return True
40
41 def getAnswer(board,col):
42     if col == n:
43         # print(s)
44         printSolution(board)
45         return
46
47     for row in range(0,n):
48         if isSafe(row,col):
49             board[row][col] = 'Q'
50             getAnswer(board,col + 1)
51             board[row][col] = '.'
52
53 def solveNQueens(n):
54     board = [['.' for _ in range(n)]]
55     getAnswer(board,0)
56
57
58 if __name__ == '__main__':
59     n = int(input("Enter number of queens"))
60     print("Different possible positions with given " + str(n) + " queens are :")
61     count = 0
62     solveNQueens(n)
```

RA1911026010033/nQuee.x

Stop

Go

RA1911026010033/nQueen

Runner: Python 3

CWD

ENV

Enter number of queens 4
Different possible positions with given 4 queens are :
[. . . Q]
[Q . . .]
[. . . Q]
[. . . .]
[. . . .]
[. . . Q]
[Q . . .]
[. . . .]
[Q . . .]
[. . . .]

Process exited with code: 0

N = 6

[illegible]

Result: The n-queens problem has been solved using a backtracking algorithm.