

**AIM:** To solve the 4 queens problem using the Backtracking technique.

**ALGORITHM:** Backtracking algorithm

- 1) Start in the leftmost column.
- 2) If all queens are placed  
    return true
- 3) Try all rows in the current column.  
    Perform the following operations for every tried row.
  - 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.

**PROGRAM :**

```
global N  
N = 4
```

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

```
def isSafe(board, row, col):

    for i in range(col):
        if board[row][i] == 1:
            return False
    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    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):

    if col >= N:
        return True

    for i in range(N):

        if isSafe(board, i, col):

            board[i][col] = 1

            if solveNQUtil(board, col + 1) == True:
                return True
```

```
        board[i][col] = 0

    return False

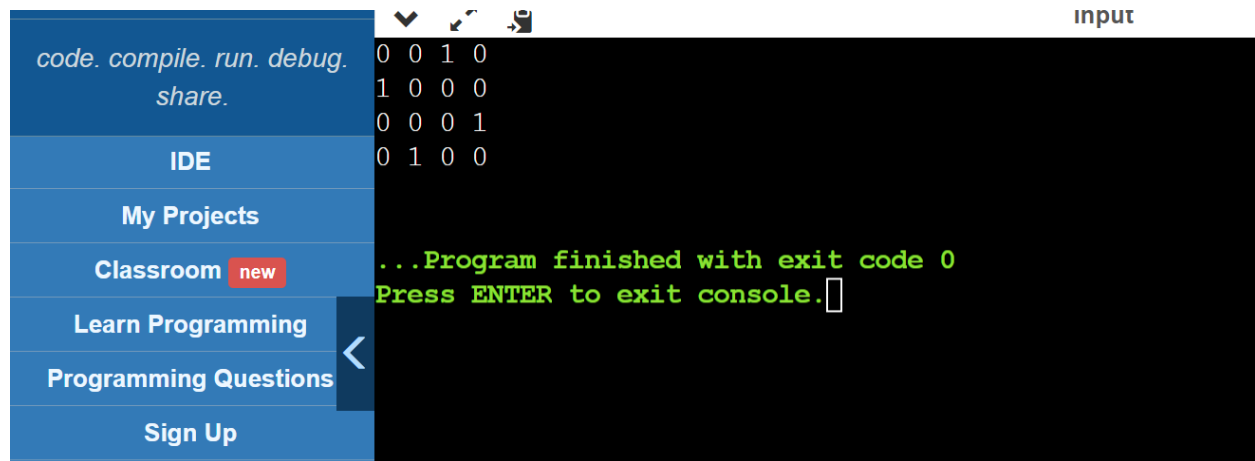
def solveNQ():
    board = [ [0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

solveNQ()
```

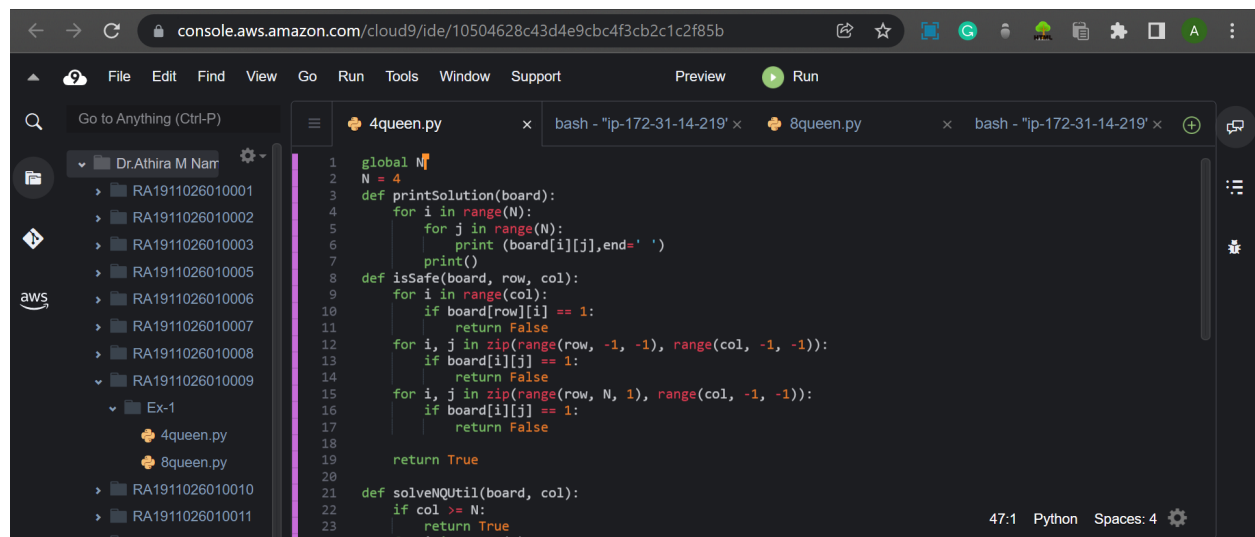
## SCREENSHOT OF THE OUTPUT:



The screenshot shows a web-based IDE interface. On the left is a vertical menu with options: `code. compile. run. debug. share.`, **IDE**, **My Projects**, **Classroom** (with a red 'new' badge), **Learn Programming**, **Programming Questions**, and **Sign Up**. On the right is a dark console area. The top right of the console has the word 'input'. The console output shows a 4x4 grid of numbers: `0 0 1 0`, `1 0 0 0`, `0 0 0 1`, and `0 1 0 0`. Below this, it says `...Program finished with exit code 0` and `Press ENTER to exit console.` with a cursor.

## OUTPUTS IN AWS :

## 4-QUEENS SOURCE CODE SCREENSHOT:



The screenshot shows the AWS Cloud9 IDE interface. The browser address bar shows `console.aws.amazon.com/cloud9/ide/10504628c43d4e9cbc4f3cb2c1c2f85b`. The IDE has a menu bar (File, Edit, Find, View, Go, Run, Tools, Window, Support) and a toolbar. The left sidebar shows a file explorer with a project named 'Dr.Athira M Nam' containing several folders (RA1911026010001 to RA1911026010011) and a file 'Ex-1' containing '4queen.py' and '8queen.py'. The main editor window shows the source code for '4queen.py'. The code is as follows:

```
1 global N
2 N = 4
3 def printSolution(board):
4     for i in range(N):
5         for j in range(N):
6             print (board[i][j],end=' ')
7             print()
8 def isSafe(board, row, col):
9     for i in range(col):
10         if board[row][i] == 1:
11             return False
12     for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
13         if board[i][j] == 1:
14             return False
15     for i, j in zip(range(row, N, 1), range(col, -1, -1)):
16         if board[i][j] == 1:
17             return False
18     return True
19
20 def solveNQueen(board, col):
21     if col >= N:
22         return True
```

The status bar at the bottom right shows '47:1 Python Spaces: 4'.

```
21 def solveNQueenUtil(board, col):
22     if col >= N:
23         return True
24     for i in range(N):
25
26         if isSafe(board, i, col):
27             board[i][col] = 1
28             if solveNQueenUtil(board, col + 1) == True:
29                 return True
30             board[i][col] = 0
31     return False
32
33 def solveNQueen():
34     board = [
35         [0, 0, 0, 0],
36         [0, 0, 0, 0],
37         [0, 0, 0, 0],
38         [0, 0, 0, 0]
39     ]
40     if solveNQueenUtil(board, 0) == False:
41         print("Solution does not exist")
42         return False
43     printSolution(board)
44     return True
45
46 solveNQueen()
47
```

OUTPUT OF 4-QUEEN:

```
Mathira:~/environment/RA1911026010009/Ex-1 $ python 4queen.py
File "4queen.py", line 10
    print (board[i][j],end= ' ')
    ^
SyntaxError: invalid syntax

Mathira:~/environment/RA1911026010009/Ex-1 $ clear
Mathira:~/environment/RA1911026010009/Ex-1 $ python3 4queen.py
0 0 0 1
1 0 0 0
0 0 0 1
0 1 0 0
Mathira:~/environment/RA1911026010009/Ex-1 $
```

8-QUEEN SOURCE CODE SCREENSHOT:

```
1 print ("Enter the number of queens")
2 N = int(input())
3 board = [[0]*N for _ in range(N)]
4 def attack(i, j):
5     for k in range(0,N):
6         if board[i][k]==1 or board[k][j]==1:
7             return True
8     for k in range(0,N):
9         for l in range(0,N):
10            if (k+l==i+j) or (k-l==i-j):
11                if board[k][l]==1:
12                    return True
13    return False
14 def N_queens(n):
15     if n==0:
16         return True
17     for i in range(0,N):
18         for j in range(0,N):
19             if (not(attack(i,j))) and (board[i][j]!=1):
20                 board[i][j] = 1
21                 if N_queens(n-1)==True:
22                     return True
23                 board[i][j] = 0
24     return False
25 N_queens(N)
26 for i in board:
27     print (i)
```

OUTPUT OF 8-QUEEN PROBLEM:

The screenshot shows a code editor with a dark theme. On the left, a file explorer displays a directory structure under 'Dr.Athira M Nam'. It contains several subdirectories named 'RA1911026010001' through 'RA1911026010009', and an 'Ex-1' directory. Inside 'Ex-1', there are two Python files: '4queen.py' and '8queen.py'. The main editor area shows a terminal window with the following content:

```
[0, 0, 1, 0, 0, 0, 0, 0]
Mathira:~/environment/RA1911026010009/Ex-1 $ python 8queen.py
Enter the number of queens
8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
Mathira:~/environment/RA1911026010009/Ex-1 $
```

**RESULT:** Thus, the implementation of 4 queens and 8-queens has been done using the Backtracking algorithm.