

Experiment No – 04

Aim: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

Source Code:

N = 8

```
def printSolution(board):
```

```
    for i in range(N):
```

```
        for j in range(N):
```

```
            print(board[i][j], end = " ")
```

```
        print()
```

```
def isSafe(row, col, slashCode, backslashCode,
```

```
        rowLookup, slashCodeLookup,
```

```
        backslashCodeLookup):
```

```
    if (slashCodeLookup[slashCode[row][col]] or
```

```
        backslashCodeLookup[backslashCode[row][col]] or
```

```
        rowLookup[row]):
```

```
        return False
```

```
    return True
```

```
def solveNQueensUtil(board, col, slashCode, backslashCode,
```

```
        rowLookup, slashCodeLookup,
```

```
        backslashCodeLookup):
```

```
    if(col >= N):
```

```
        return True
```

```
    for i in range(N):
```

```
        if(isSafe(i, col, slashCode, backslashCode,
```

```
            rowLookup, slashCodeLookup,
```

```
            backslashCodeLookup)):
```

```
            """ Place this queen in board[i][col] """
```

```
            board[i][col] = 1
```

```
            rowLookup[i] = True
```

```
            slashCodeLookup[slashCode[i][col]] = True
```

```

backslashCodeLookup[backslashCode[i][col]] = True

    if(solveNQueensUtil(board, col + 1,
                        slashCode, backslashCode,
                        rowLookup, slashCodeLookup,
                        backslashCodeLookup)):
        return True

    """ Remove queen from board[i][col] """
    board[i][col] = 0
    rowLookup[i] = False
    slashCodeLookup[slashCode[i][col]] = False
    backslashCodeLookup[backslashCode[i][col]] = False

    """ If queen can not be place in any row in
    this column col then return False """
    return False

def solveNQueens():
    board = [[0 for i in range(N)]
              for j in range(N)]
    # helper matrices
    slashCode = [[0 for i in range(N)]
                  for j in range(N)]
    backslashCode = [[0 for i in range(N)]
                      for j in range(N)]
    # arrays to tell us which rows are occupied
    rowLookup = [False] * N
    x = 2 * N - 1
    slashCodeLookup = [False] * x
    backslashCodeLookup = [False] * x
    # initialize helper matrices
    for rr in range(N):
        for cc in range(N):

```

```

        slashCode[rr][cc] = rr + cc

        backslashCode[rr][cc] = rr - cc + 7

    if(solveNQueensUtil(board, 0, slashCode, backslashCode,
                        rowLookup, slashCodeLookup,
                        backslashCodeLookup) == False):

        print("Solution does not exist")

    return False

# solution found
printSolution(board)

return True

# Driver Code
solveNQueens()

```

Output:

```

1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

```

True

The screenshot shows a JupyterLab window titled 'Untitled6' with a last checkpoint of '1 minute ago'. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and code execution. The code cell contains the same Python code as shown in the previous blocks. The output area displays the solution board as a grid of 0s and 1s, followed by the prompt '[3]: True'.