In [ ]:
```
N-Queen using Backtracking Algorithm
IS-ATTACK(i, j, board, N)
  // checking in the column j
  for k in 1 to i-1
    if board[k][j]==1
      return TRUE

  // checking upper right diagonal
  k = i-1
  l = j+1
  while k>=1 and l<=N
    if board[k][l] == 1
      return TRUE
    k=k+1
    l=l+1

  // checking upper left diagonal
  k = i-1
  l = j-1
  while k>=1 and l>=1
    if board[k][l] == 1
      return TRUE
    k=k-1
    l=l-1

  return FALSE

N-QUEEN(row, n, N, board)
  if n==0
    return TRUE

  for j in 1 to N
    if !IS-ATTACK(row, j, board, N)
      board[row][j] = 1

      if N-QUEEN(row+1, n-1, N, board)
        return TRUE

      board[row][j] = 0 //backtracking, changing current decision
  return FALSE
```

In [2]:
```
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):

        # 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):

        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

# driver program to test above function
solveNQ()
```

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
True
```

Out[2]: