```
# 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()


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, 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):
            # Place this queen in board[i][col]
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solveNQUtil(board, col + 1):
                return True

            # If placing queen doesn't lead to a solution, backtrack
            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 not solveNQUtil(board, 0):
        print("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver code
solveNQ()
```

المشكلةN-Queens Problem :

المطلوب هو وضع NNNملكة على رقعة شطرنج N×NN \times NN×N، بحيث:

1. لا تهاجم أي ملكة الأخرى.
2. لا تهاجم الملكة أي ملكة أخرى على نفس:
   - الصف.
   - العمود.
   - القطر الرئيسي.(diagonal)
   - القطر الثانوي.(anti-diagonal)

الكود

1.تعريف المتغير N

```
python
Copy code
global N
N = 4
```

- Nهو عدد الملكات وحجم الرقعة. في هذا المثال، N=4N = 4N=4مما يعني رقعة شطرنج .4×44 \times 44×4

2.دالة printSolution(board)

```
python
Copy code
def printSolution(board):
for i in range(N):
for j in range(N):
print(board[i][j], end=' ')
print()
```

- **وظيفتها**:
  - تقوم بطباعة الحل النهائي (أي توزيع الملكات على الرقعة).
  - تأخذ مصفوفة boardكمدخل.
- **كيفية عملها**:
  - تمر عبر كل صف من المصفوفة.
  - تطبع القيم الموجودة في كل خانة.
  - القيم في المصفوفة:
    - :1تشير إلى وجود ملكة في تلك الخانة.
    - :0تشير إلى أن الخانة فارغة.

---

3. دالة isSafe(board, row, col)

```
python
Copy code
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
```

- **وظيفتها**:
  - تتحقق إذا كان يمكن وضع ملكة بأمان في الموقع .(row, col)
- **كيف تعمل**:
  - **التحقق من الصف على اليسار**:
    - تمر عبر جميع الأعمدة على يسار العمود الحالي.(col)
    - إذا وجدت ملكة في أي عمود، تعيد .False
  - **التحقق من القطر العلوي الأيسر**:
    - تستخدم zip للمرور عبر الخلايا في القطر العلوي الأيسر.
    - إذا وجدت ملكة، تعيد .False
  - **التحقق من القطر السفلي الأيسر**:
    - تستخدم zip للمرور عبر الخلايا في القطر السفلي الأيسر.
    - إذا وجدت ملكة، تعيد .False
  - إذا لم تجد أي تعارضات، تعيد .True

---

4. دالة solveNQUtil(board, col)

```
python
Copy code
def solveNQUtil(board, col):
# Base case: If all queens are placed, 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):
# Place this queen in board[i][col]
board[i][col] = 1

# Recur to place the rest of the queens
if solveNQUtil(board, col + 1):
return True

# If placing queen doesn't lead to a solution, backtrack
board[i][col] = 0

return False
```

- **وظيفتها**:
  - تقوم بوضع الملكات باستخدام تقنية **Backtracking**.
- **كيف تعمل**:
  - **الحالة الأساسية**:
    - إذا تم وضع جميع الملكات بنجاح(col >= N) ، تعيد True.
  - **التجربة مع كل صف**:
    - تحاول وضع الملكة في كل صف داخل العمود الحالي.(col)
    - تستدعي isSafe للتحقق إذا كان الوضع آمنًا.
    - إذا كان الوضع آمنًا:
      - تضع الملكة.(board[i][col] = 1)
      - تستدعي نفسها بشكل عودي للانتقال إلى العمود التالي.
      - إذا وجدت حلاً، تعيد True.
    - إذا لم يؤد وضع الملكة إلى حل:
      - تُزيل الملكة (board[i][col] = 0) وتتابع التجربة (عملية **Backtracking**).
  - **إذا لم تنجح المحاولات في أي صف**:
    - تعيد False.

---

5. دالة solveNQ()

```
python
Copy code
def solveNQ():
board = [ [0, 0, 0, 0],
[0, 0, 0, 0],
[0, 0, 0, 0],
[0, 0, 0, 0]
]

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

printSolution(board)
```

return True

- **وظيفتها**:
  - نقطة الدخول الرئيسية لحل المشكلة.
  - تُجهز الرقعة الفارغة.(board)
  - تستدعي solveNQUtilلبدء وضع الملكات.
  - تطبع النتيجة إذا كان هناك حل، أو رسالة إذا لم يكن هناك حل.

6.التشغيل(Driver Code)

python
Copy code
solveNQ()

- يقوم بتشغيل دالة solveNQ()لبدء حل المشكلة.

كيف يعمل الكود بشكل كامل؟

1. يبدأ بتجهيز رقعة شطرنج فارغة.
2. يبدأ من العمود الأول (العمود 0).
3. يحاول وضع ملكة في كل صف من العمود الحالي.
4. يتحقق إذا كان الوضع آمنًا باستخدام isSafe.
5. إذا كان الوضع آمنًا:
   - يضع الملكة ويكمل البحث في الأعمدة التالية.
6. إذا لم يتمكن من إيجاد حل، يعود خطوة إلى الوراء (**Backtrack**) ويجرب صفوفًا أخرى.
7. إذا أكمل جميع الأعمدة بنجاح:
   - يطبع الحل.
8. إذا لم يجد حلًا بعد تجربة جميع الاحتمالات:
   - يطبع."Solution does not exist"