Implement a solution for a Constraint Satisfaction Problem using Backtracking for n-queens problem.

```
def solveNQueens(n: int):
    res = []
    board = [['.' for _ in range(n)] for _ in range(n)]

    def isSafe(row, col):
        for i in range(row):
            if board[i][col] == 'Q':
                return False
        # Check upper-left diagonal
        i, j = row, col
        while i >= 0 and j >= 0:
            if board[i][j] == 'Q':
                return False
            i -= 1
            j -= 1
        # Check upper-right diagonal
        i, j = row, col
        while i >= 0 and j < n:
            if board[i][j] == 'Q':
                return False
            i -= 1
            j += 1
        return True

    def backtrack(row):
        if row == n:
            res.append([''.join(r) for r in board])
            return
        for col in range(n):
            if isSafe(row, col):
                board[row][col] = 'Q'
                backtrack(row + 1)
```

```
            board[row][col] = '.'  # backtrack

    backtrack(0)
    return res

def printSolutions(boards):
    for idx, board in enumerate(boards):
        print(f"Solution {idx+1}")
        for row in board:
            print(" ".join(row))
        print()

if __name__ == "__main__":
    boards = solveNQueens(4)
    printSolutions(boards)
## OUTPUT
Solution 1
. Q . .
. . . Q
Q . . .
. . Q .

Solution 2
. . Q .
Q . . .
. . . Q
. Q . .
```

Implement a solution for a Constraint Satisfaction Problem using Branch and Bound for n-queens problem.

```
def solveNQueens(n : int):
    col = set()
    posDiag = set() # determined by r+c
    negDiag = set() # determined by r-c

    res = []
    board = [['.' for _ in range(n)] for _ in range(n)]

    def backtrack(r):
        if (r == n):
            res.append([''.join(row) for row in board])
            return
        for c in range(n):
            if (c in col or r+c in posDiag or r-c in negDiag):
                continue
            col.add(c)
            posDiag.add(r + c)
            negDiag.add(r - c)
            board[r][c] = 'Q'
            backtrack(r + 1)
            col.remove(c)
            posDiag.remove(r + c)
            negDiag.remove(r - c)
            board[r][c] = '.'
    backtrack(0)
    return res

def printSolutions(boards):
    for board in enumerate(boards):
        print(f"Solution: {board[0]+1}")
        for row in board[1]:
            for col in row:
                print(col, end=' ')
```

```
            print()
        print()

if __name__ == "__main__":
    boards = solveNQueens(8)
    printSolutions(boards)
RUN
boards = solveNQueens(4)
## OUTPUT
Solution: 1
. Q . .
. . . Q
Q . . .
. . Q .

Solution: 2
. . Q .
Q . . .
. . . Q
. Q . .
```