Group 13 Term project N queen

2020/12/11

Instructor: Ilias S. Kotsireas

Hongshan Shang	**163042150**
Dekai Meng	**186800570**
Aozhou Hao	**183152440**
Hao Wu	**166802630**
XiaoHu He	**160142400**

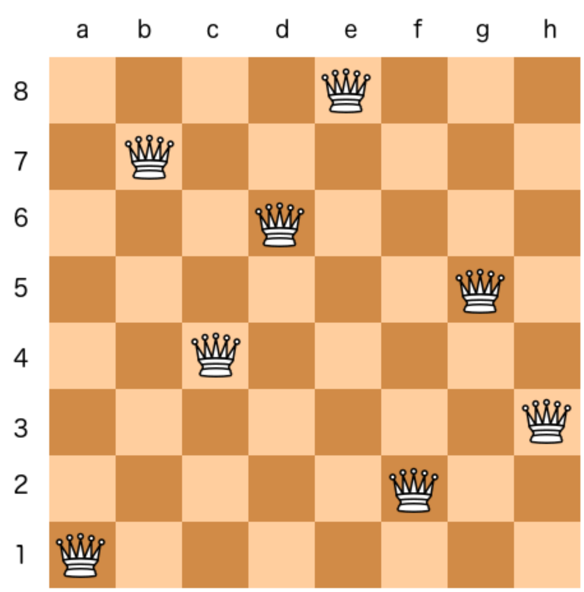


Table of contents:

1, Introduction

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. To be specific, none of two queens in the chessboard can be on the same row, column and diagonal line.

2, Algorithm

We had used min conflicts to solve the CSPS, which had been discussed in class. We had an initial state, every column had only one queen, but their row may be the same. Firstly, the algorithm searches on which the same column moves for the number of conflicts. If two queens would attack from the same direction, then the conflict is only counted one time. That will choose the minimal conflict value for each variable, if a queen is in a state of minimum conflict, it will not move.Then, the algorithm moves the queen to the square with the minimum number of conflicts. We repeated the process above until we got the minimum number of conflicts with 0. If we cannot get 0 which over the max step, that will be stopped.

3, The code is

```
import time

from copy import deepcopy

from random import choice

import numpy as np

import matplotlib

import matplotlib.pyplot as plt


Step = 0



class Board:

    """

    ------------------------------------------------------

    A Board object that keeps track of the conflicting queens and can update
their constraints.

    ------------------------------------------------------

    """


    def __init__(self, n):
```

```
        self._n = n
        self._queen_rows = {i: set() for i in range(1, self._n + 1)}
        self._queen_posdiag = {i: set() for i in range(1, 2 * self._n)}
        self._queen_negdiag = {i: set() for i in range(1, 2 * self._n)}

    def set_queen(self, x, y, constraints):
        """
        -------------------------------------------------------
        Set a queen on the board.
        -------------------------------------------------------
        args:
            x:the x value on the board
            y:the y value on the board
            constraints:the dict of conflicts for each queen
        -------------------------------------------------------
        """
        # get the combined set of conflicted queens
        combined = self._queen_rows[y] | self._queen_posdiag[y + (x - 1)] |
self._queen_negdiag[y + (self._n - x)]

        # update the number of conflicts for each queen by 1
        for i in combined:
            constraints[i] += 1

        # add the queen to the board
        self._queen_rows[y].add(x)
        self._queen_posdiag[y + (x - 1)].add(x)
        self._queen_negdiag[y + (self._n - x)].add(x)

        # update number of conflicts
        constraints[x] = len(combined)
```

```
        return

    def remove_queen(self, x, y, constraints):
        """
        -------------------------------------------------------
        Removes a queen on the board.
        -------------------------------------------------------
        args:
            x:the x value on the board
            y:the y value on the board
            constraints:the dict of conflicts for each
                queen
        -------------------------------------------------------
        """
        # get the combined set of conflicted queens
        combined = self._queen_rows[y] | self._queen_posdiag[y + (x - 1)] |
self._queen_negdiag[y + (self._n - x)]

        # update the number of conflicts for each queen by 1
        for i in combined:
            constraints[i] -= 1

        # removes the queen from the board
        self._queen_rows[y].remove(x)
        self._queen_posdiag[y + (x - 1)].remove(x)
        self._queen_negdiag[y + (self._n - x)].remove(x)

        # update the number of conflicts
        constraints[x] = 0
        return
```

```
    def get_num_conflicts(self, x, y):
        """
        -------------------------------------------------------
        Get the number of conflicts for a point on the
        board.
        -------------------------------------------------------
        args:
            x:the x value on the board
            y:the y value on the board
        Returns:
            c:the number of conflicted queens
        -------------------------------------------------------
        """
        # get the combined set of conflicted queens
        combined = self._queen_rows[y] | self._queen_posdiag[y + (x - 1)] |
self._queen_negdiag[y + (self._n - x)]

        return len(combined)


class CSP:
    """
    -------------------------------------------------------
    A CSP object that holds the variables, domains and
    constraints.
    -------------------------------------------------------
    """

    def __init__(self, variables, domains, constraints):
        self.variables = variables
        self.domains = domains
```

```
        self.constraints = constraints


# FUNCTIONS #
def min_conflicts(csp, n, board, max_steps=100):
    """
    ------------------------------------------------------
    Min-Conflicts algorithm for solving CSPs by
    local search.
    ------------------------------------------------------
    csp:a CSP with components(X, D, C)
    n:the number of queens
    board:the board object keeping track of the
        queens in conflict
    max_steps:the number of steps allowed before
        giving up
    Returns:
        A solution or failure (False)
    ------------------------------------------------------
    """
    # Tabu Search list and variable to avoid repeating moves
    past_var = {}
    past_queen = None

    # sets the size of the tabu list
    x = 50 if n >= 100 else (n // 2)

    for i in range(1, max_steps + 1):

        # get the list of conflcited queens from the csp constraints
```

```
        conflicted = [i for i, j in csp.constraints.items() if j != 0]

        # if there are not more conflicting queens then the problem is sovled
        if not conflicted:
            # print("Steps: {}".format(i))
            global Step
            Step = i
            return csp

        # remove the past queen from the search space
        if past_queen is not None and past_queen in conflicted:
            conflicted.remove(past_queen)

        # get a random queen from the conflicted list and remove it from the board
        var = choice(conflicted)
        past_queen = var
        board.remove_queen(var, csp.domains[var], csp.constraints)

        # set the queens position into the tabu list so we dont place it back here
        if var in past_var:
            if csp.domains[var] not in past_var[var]:
                past_var[var].append(csp.domains[var])
        else:
            past_var[var] = [csp.domains[var]]

        # get the position with the least conflicts
        value = conflicts(var, csp.domains[var], n, csp, past_var[var], board)
        if len(past_var[var]) >= x:
            past_var[var].pop(0)
```

```
        # set the queen back on the board
        csp.domains[var] = value
        board.set_queen(var, value, csp.constraints)

        # print("n - len(csp.constraints):", n - len(conflicted))
        if i % int(max_steps/5) == 0:
            print("Step:", i)
            print_time()
    return False


def get_least_conflicts_y(x, n, possible, board):
    """
    -------------------------------------------------------
    Get's the position with the least conflicts for the
    column.
    -------------------------------------------------------
    x:the x value on the board
    n:the number of queens
    possible:the list of possible moves
    board:the board object keeping track of the queens in conflict
    Returns:
        y - the y value on the board
    -------------------------------------------------------
    """
    # the list of y values that have the least conflicts
    conflict_list, min_count = [possible[0]], board.get_num_conflicts(x,
possible[0])

    # for the rest of the column find the positions with the least conflicts
```

```
    for i in possible[1:]:
        count = board.get_num_conflicts(x, i)
        # update the min_count and list
        if min_count > count:
            min_count = count
            conflict_list = [i]
        elif min_count == count:
            conflict_list.append(i)

    return choice(conflict_list)


def conflicts(var, v, n, csp, not_possible, board):
    """
    -------------------------------------------------------
    Get's the position with the least conflicts for the
    column.
    -------------------------------------------------------
    var:the x value on the board (or queen)
    v:the current y value
    n:the number of queens
    csp:a CSP with components(X, D, C)
    not_possible: the tabu list for that column
    board: the board object keeping track of thequeens in conflict
    Returns:
        y - the y value on the board
    -------------------------------------------------------
    """
    x, y = var, v
    conflict_list, min_count = [], None
```

```
    # check the column for the position with the least conflicts
    for i in range(1, n + 1):
        # skip the position we just came from
        if i == y:
            continue
        count = board.get_num_conflicts(x, i)
        if min_count is not None and min_count > count:
            min_count = count
            conflict_list = [i]
        elif min_count is not None and min_count == count:
            conflict_list.append(i)
        elif min_count is None:
            min_count = count
            conflict_list = [i]

    clist = list(set(conflict_list) - set(not_possible))

    # if the conflict is has positions that the queen has not been to yet
    if clist:
        return choice(clist)

    # if there were no positions available for the queen choose a random one
    return choice(conflict_list)


# functions to create and print the board
def create_board(n):
    return [["-" for i in range(n)] for j in range(n)]
```

```
def print_time():
    global start_time
    print(" ⏱ {:.5f} secs".format(time.time()-start_time))



def print_board(name, assignment):
    assignment_list = [[0 for _ in range(len(assignment))] for _ in
range(len(assignment))]
    # print(len(assignment_list), len(assignment), n)
    for i in range(len(assignment)):
        for j in range(len(assignment)):
            if (i + j) % 2 == 0:
                assignment_list[i][j] = 1

    for key in assignment:
        # print("assignment[key]:", assignment[key])
        # print("key:", key)
        assignment_list[assignment[key] - 1][key - 1] = 2

    for i in range(len(assignment_list)):
        print('|', end='')
        for k in range(len(assignment_list)):
            if assignment_list[i][k] == 2:
                print('Q', end='')
            else:
                print('■', end='')
            print('|', end='')
        print()
    white2 = (1, 1, 1)
    white1 = (0.93, 0.93, 0.93)
```

```
    yellow2 = (255 / 255, 165 / 255, 0 / 255)

    gray2 = (0.96, 0.96, 0.96)

    gray1 = (230 / 255, 230 / 255, 230 / 255)

    yellow1 = (247 / 255, 220 / 255, 111 / 255)

    black = (0, 0, 0)

    my_cmap = matplotlib.colors.LinearSegmentedColormap.from_list('', [gray1,
white1, yellow1])

    cs = plt.imshow(assignment_list, cmap=my_cmap)


    plt.xticks(np.linspace(0, n, n, endpoint=False))

    plt.yticks(np.linspace(0, n, n, endpoint=False))

    plt.tick_params(bottom=False, left=False, labeltop=False,
labelbottom=False, labelleft=False, labelright=False)

    plt.title(name, fontweight="bold")

    plt.show()

    return



# initialize the number of queens

n = 10

# initialize the max_steps

max_steps = 100

while n <= 1000000:

    print("------------- Start -------------")

    print("Number of Queens:", n)

    print("Max step:", max_steps)

    start_time = time.time()

    # create the board object that keeps track of the queen placements and
conflicts

    board = Board(n)

    # variables and _vars are a list of x and y values basically

    variables = [i for i in range(1, n + 1)]
```

```
_var = deepcopy(variables)
# initialize the python dict of the number of conflicts for each queen
constraints = {i: 0 for i in variables}

# place the initial queen on the board randomly
y = choice(_var)
# initialize the list of domains (x->y)
domains = {1: y}
_var.remove(y)
# place the queen on the board and update the number of constraints
board.set_queen(1, y, constraints)
# place the rest of the queens
# for i in range(2, n + 1):
#     y = get_least_conflicts_y(i, n, _var, board)
#     domains[i] = y
#     board.set_queen(i, y, constraints)
#     _var.remove(y)
#     if i % int(n/5) == 0:
#         print_time()

for i in range(2, n + 1):
    y = choice(range(1, n + 1))
    domains[i] = y
    board.set_queen(i, y, constraints)

# initialize the CSP
csp = CSP(variables, domains, constraints)

print("Set-up Time: {:0.5f} secs".format(time.time() - start_time))
```

```
# print the inital board for smaller boards
if n <= 80:
    print()
    print("Initial Board")
    b = create_board(n)
    for key, value in csp.domains.items():
        if constraints[key] > 0:
            b[value - 1][key - 1] = "Q"
        else:
            b[value - 1][key - 1] = "Q"

    for i in range(len(b)):
        print('|', end='')
        for j in range(len(b)):
            if b[i][j] == 'Q':
                print('Q', end='')
            else:
                print('■', end='')
            print('|', end='')
        print()

# to calculate the solve time
solve_time = time.time()
print("Solving...")
# min_conflicts
assignment = min_conflicts(csp, n, board, max_steps)

# if min-conflicts solved the problem
if assignment:
    # show times
```

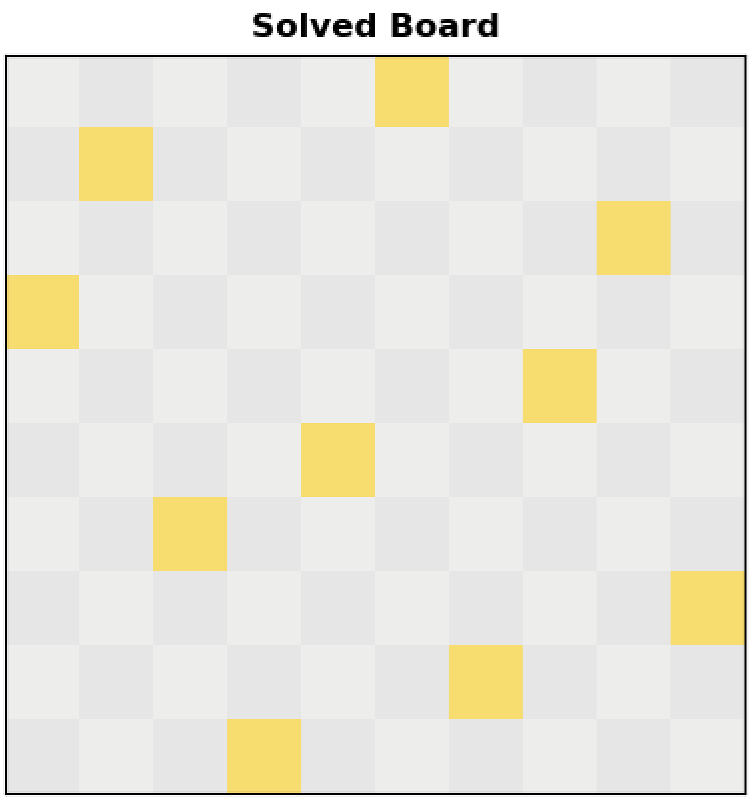
```
    end_time = time.time()
    # print the board if the board is small
    if n <= 80:
        print()
        print("Solved Board")
        print_board("Solved Board", assignment.domains)
    elif n <= 100:
        # or print to an output.txt file if we're computing larger numbers
        file_name = "output" + str(n) + ".txt"
        with open(file_name, "w") as f:
            print("Number of Queens:", n, file=f)
            b = create_board(n)
            for key, value in assignment.domains.items():
                b[value - 1][key - 1] = "Q"
            for i in range(len(b)):
                print('|', end='', file=f)
                for k in range(len(b)):
                    if b[i][k] == 'Q':
                        print('Q', end='', file=f)
                    else:
                        print('■', end='', file=f)
                    print('|', end='', file=f)
                print(file=f)
            print('\n\n', file=f)

    print("Steps: {}".format(Step))
    print("Solve Time: {:0.5f} secs".format(end_time - solve_time))
    print("Total Time: {:0.5f} secs".format(end_time - start_time))
    print("-------------- End --------------\n\n")
```

```
        if n < 10000:
            n *= 10
        elif n == 10000:
            n = 50000
            max_steps *= 10
        elif n == 50000:
            n = 100000
            max_steps *= 10
        elif n <= 10000000:
            n += 100000
            max_steps *= 10
        else:
            break

    else:
        # the min-conflicts algorithm failed to solve the csp in the max
steps allowed
        pre_max = max_steps
        max_steps *= 100
        print("Increase Max Steps({}) to {}.".format(pre_max, max_steps))
        print("------------ Continue -----------")
```

4. Input and Output

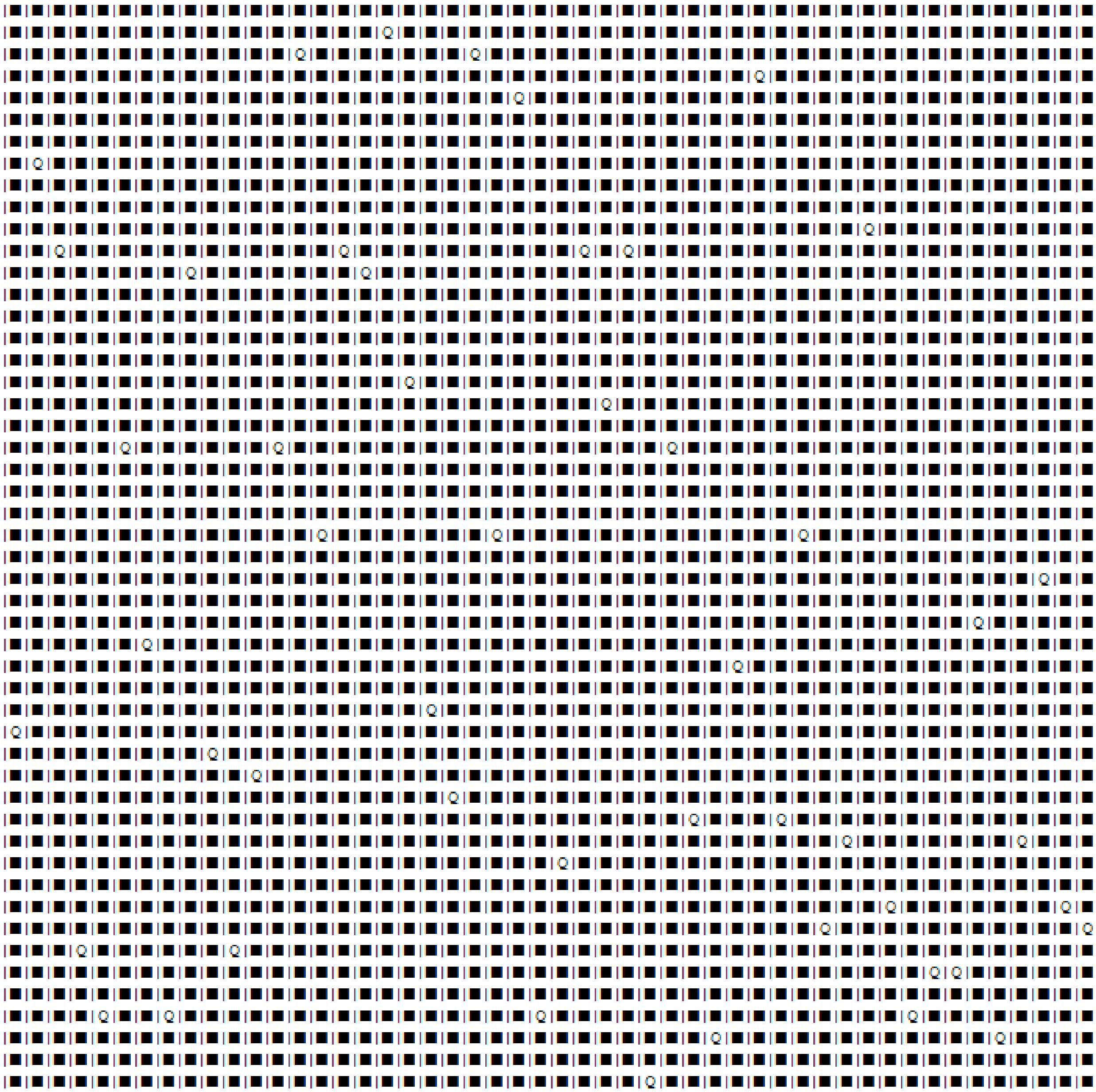
When N=10 (10X10 board)

```
------------- Start -------------
Number of Queens: 10
Max step: 100
Set-up Time: 0.00000 secs

Initial Board
|■|■|■|■|■|■|■|■|■|■|
|■|■|■|Q|■|■|■|■|■|■|
|■|■|Q|■|■|■|■|■|Q|■|
|■|Q|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|Q|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|
|Q|■|■|■|■|■|■|Q|■|■|
|■|■|■|■|■|Q|Q|■|■|Q|
Solving...

Solved Board
|■|■|■|■|■|Q|■|■|■|■|
|■|Q|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|Q|■|
|Q|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|Q|■|■|
|■|■|■|■|Q|■|■|■|■|■|
|■|■|Q|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|Q|
|■|■|■|■|■|■|Q|■|■|■|
|■|■|■|Q|■|■|■|■|■|■|
Steps: 12
Solve Time: 0.00099 secs
Total Time: 0.00200 secs
-------------- End --------------
```



When N=50 (50X50 board)

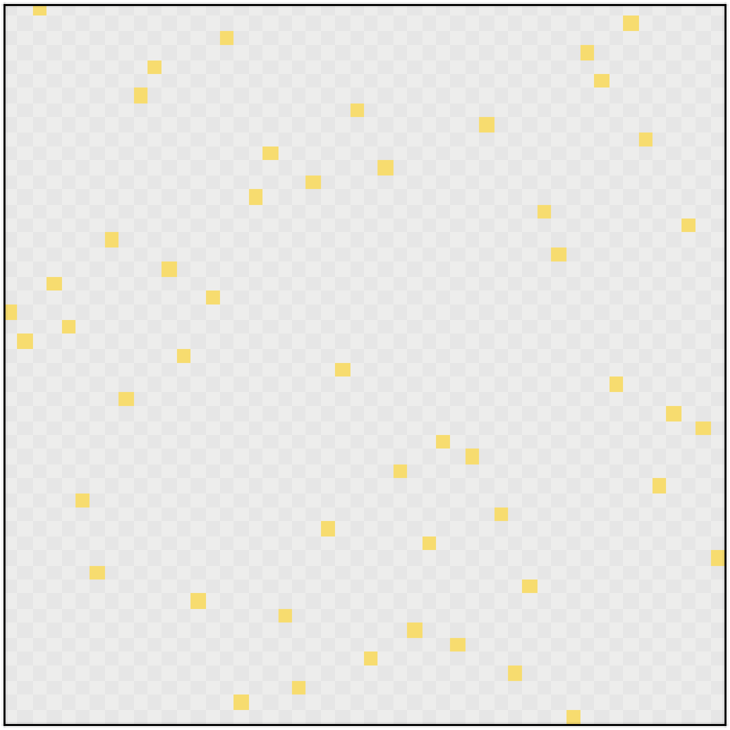
```
------------- Start -------------
Number of Queens: 50
Max step: 100
Set-up Time: 0.00000 secs

Initial Board
```



```
Solving...
Step: 20
⏱ 0.02399 secs
Step: 40
⏱ 0.02698 secs
Step: 60
⏱ 0.02998 secs

Solved Board
```

```
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|
|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|Q|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|
```

Solved Board



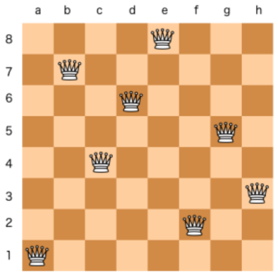
```
Steps: 70
Solve Time: 0.00899 secs
Total Time: 0.03098 secs
-------------- End --------------
```

Group 13 N queen
2020/12/11
Instructor: Ilias S. Kotsireas

Hongshan Shang， Hao wu
Dekai Meng， Aozhou Hao
Xioahu He

Introduction

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. To be specific, none of two queens in the chessboard can be on the same row, column and diagonal line.



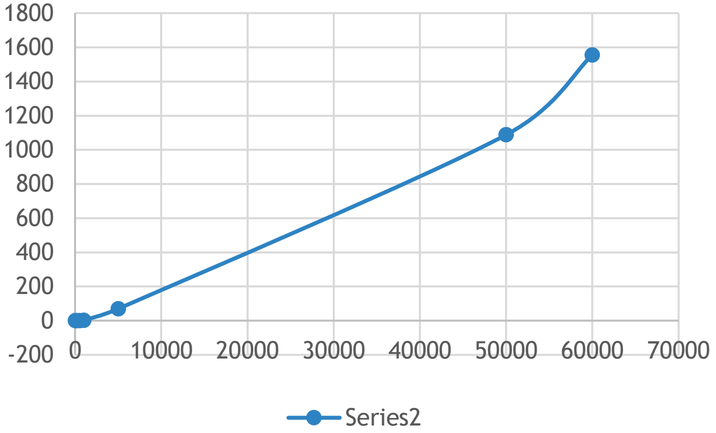
Algorithm

Local search algorithm: Min conflicts

We had an initial state, every column had only one queen, may with the flexible row. Then the algorithm will search on which the same column moves for the number of conflicts. If two queens would attack from the same row, column and diagonal line, then the conflict is only counted one time. That will choose the minimal conflict value for each variable. Then the algorithm moves the queen to the square with the minimum number of conflicts. Repeat those processes until we get the minimum number of conflicts with 0. If we cannot get 0 which over the max step, that will be stopped.

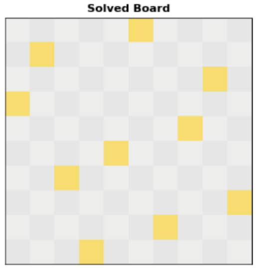
Time we used at each number of NxN board：
Our maximum limit is 100,000

N	Time(s)
10	0.00053
100	0.00599
500	0.94637
1000	2.75884
5000	69.40918
50000	1087.79748
60000	1555.67212



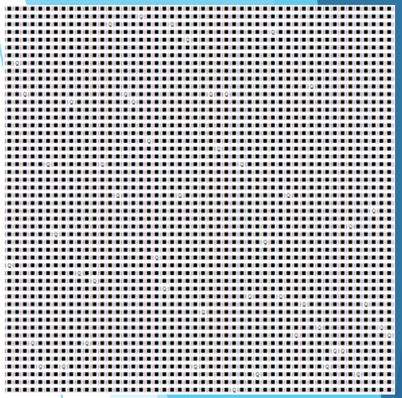
Input and Output
When N=10 (10X10 board)

```
------------- Start -------------
Number of Queens: 10
Max step: 100
Set-up Time: 0.00000 secs
Initial Board
|■|■|■|■|■|■|■|■|■|■|
|■|■|■|Q|■|■|■|■|■|■|
|■|■|Q|■|■|■|■|■|Q|■|
|■|Q|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|Q|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|■|
|Q|■|■|■|■|■|■|Q|■|■|
|■|■|■|■|■|Q|Q|■|■|Q|
Solved Board
|■|■|■|■|■|Q|■|■|■|■|
|■|Q|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|Q|■|
|Q|■|■|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|Q|■|■|
|■|■|■|■|Q|■|■|■|■|■|
|■|■|Q|■|■|■|■|■|■|■|
|■|■|■|■|■|■|■|■|■|Q|
|■|■|■|■|■|■|Q|■|■|■|
|■|■|■|Q|■|■|■|■|■|■|
Steps: 12
Solve Time: 0.00099 secs
-------------- End --------------
```

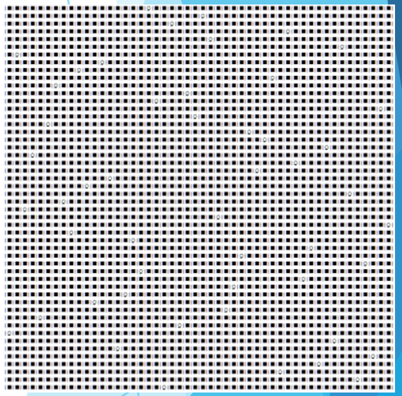


When N=50 (50X50 board)

------------- Start -------------
Number of Queens: 10
Initial Board



Solved Board



step： 70
Solve Time: 0.00899 secs