Assignment 1

Artificial Intelligence

CISC 352, Winter, 2019

Due Friday, February 15, 2019 before Midnight

Preamble

N-Queen's Problem

Chess composer Max Bezzel published the eight queens puzzle in 1848. The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. Franz Nauck published the first solutions in 1850. Nauck also extended the puzzle to the *n* queens problem, with *n* queens placed on a chessboard of nxn squares.

Since then, many mathematicians, including Carl Gauss, have worked on both the eight queens puzzle and its generalized *n*-queens version. In 1972, Edsger Dijkstra used this problem to illustrate the power of structured programming. He published a highly detailed description of a depth-first backtracking algorithm. While depth-first backtracking is guaranteed to find a solution, it is inefficient compared with an iterative repair method.

An iterative repair algorithm to solve the *n*-queens problem starts with all queens on the board, for example, with one queen per column. It then counts the number of conflicts with other queens and uses a heuristic to determine how to improve the placement of the queens. The minimum-conflicts heuristic (moving the piece with the largest number of conflicts to the square in the same column where the number of conflicts is smallest) is particularly effective at solving the n-queens problem. It finds a solution to the 1,000,000-queen problem in less than 50 steps on average.

This assumes that the initial configuration is reasonably good (e.g., if a million queens all start in the same row, it will take at least 999,999 steps to fix it). A reasonably good starting point can, for instance, be found by putting each queen in its own row and column so that it conflicts with the smallest number of queens already on the board.

Note that iterative repair, unlike backtracking, does not guarantee a solution. Like other local search methods, it may get stuck on a local optimum. A fix for this is to restart the algorithm with a different initial configuration. Despite this drawback, it can solve problem sizes that are several orders of magnitude beyond the scope of a depth-first search. And, given a random initial state, the minimum-conflicts heuristic can solve the *n*-queens problem in almost constant time for arbitrary *n* with high probability.

Solve the Following Problem

Programming Language Requirement: Python 3

Max Group Size: 6, Min Group Size: 3

1. **N-Queens Problem.**
   Implement the iterative repair method using the minimum conflicts algorithm to solve the *n*-queens problem. Name the program **nqueens.py**.

   Input.
   Your program will read input from a file called "nqueens.txt" which contains successive lines of input. Each line of input consists of a single integer value, n, where n > 3 and n <= 10,000,000, that determines the size of the n-queens problem to be solved.

   For instance, line 1 of a sample "nqueens.txt" input file could contain the integer value "128". This means that you need to solve the problem for 128 queens on a 128x128 size chessboard. Each successive line of input contains a new integer value that determines the size of the problem to solve.

   A **small sample input file** might be:
   32
   64
   128
   256
   This indicates you need to produce solutions to, first, the 32-queens problem, then the 64-queens problem, and so on.

   Output.
   Your program will write output to a single file named "nqueens_out.txt".
   Output consists of successive 1-based matrices (i.e., the queen in the first row is at location 1, not location zero) containing the position of each queen.
   4-queens example output matrix:
   [2,4,1,3]

   The **small sample input file** contains 4 input problems, so 4 matrices would be written to the output file "nqueens_out.txt", one per line.

   Note that solutions are not unique. Further note that the algorithm is not guaranteed to find a solution. You will need to restart the algorithm with a new configuration (perhaps generated randomly or through a greedy approach) and repeat until it produces a solution.

Grading

Part 1: **N-Queens Problem.**

a. Your n-queens program will be run with three input files: the first consists of small n, the second consists of medium n, and third consists of large n. You will be graded on correctness and algorithm efficiency.

$$\text{Small: } \frac{\#correctSolutions}{\#totalFormulas} \times 15 + 10 * smallEfficiency$$

$$\text{Medium: } \frac{\#correctSolutions}{\#totalFormulas} \times 15 + 10 * medEfficiency$$

$$\text{Large: } \frac{\#correctSolutions}{\#totalFormulas} \times 15 + 10 * largeEfficiency$$

smallEfficiency: (<3 min = 1, <5 min = .8, <7 min = .6, <10 min = .4, <15 min = .2) Runtime longer than 15 minutes on small *n* problems will result in a 0 for the Part 1 grade.
medEfficiency: (<5 min = 1, <10 min = .8, <15 min = .6, <20 min = .4, <25 min = .2) Runtime longer than 25 minutes on medium *n* problems will result in a 0 for the Medium and Large grades.
largeEfficiency: (<10 min = 1, <15 min = .8, <20 min = .6, <25 min = .4, <30 min = .2) Runtime longer than 30 minutes on the large *n* problems will result in a 0 for the Large grade.
largeBonus: (<5 min = 5, <3 min = 10): Note, you only get bonus if solutions are correct.

$$Part\ 1\ Grade: \sum Small + Medium + Large + largeBonus$$

b. Part 1 is worth a maximum of **75 points**, but with bonus, you can earn up to **85 points**.

Part 2: **Technical Document**

a. The second part is a grade on your technical document describing your algorithm. This is worth **25 points**. The grade will include grammar, spelling, presentation quality, and the description of your algorithms. I should be able to know how your algorithms work from reading the document. Regarding presentation quality, **do not submit a .txt file**. The document should be presented as nicely as possible, with headings where appropriate. **Put the name and student ids of all group members on this document. Convert the file to a PDF and name it nqueens.pdf.**

The total points you can earn for this assignment is **100**.

Including bonus, you can earn up to **110** points.

This assignment is **11%** of your total grade for the course.

Submit the program and technical document, one per group, compressed together in a zip file named **nqueens.zip** to the onQ Assignment 1 Dropbox.

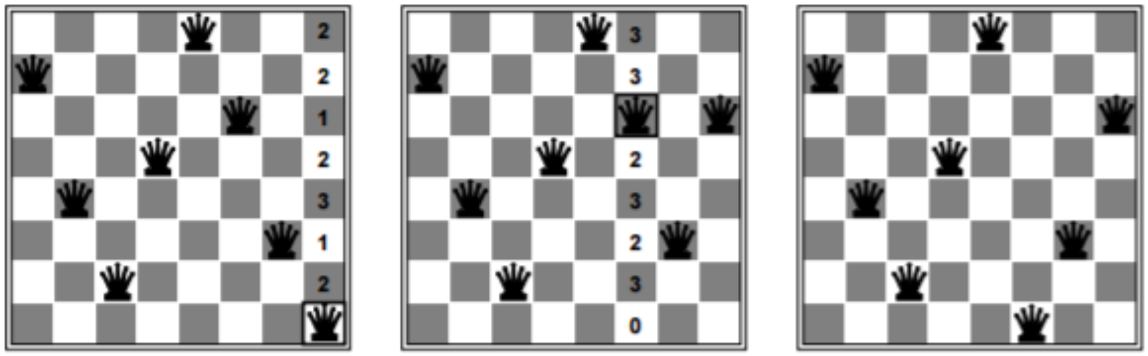
Notes

Algorithm

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
   inputs: csp, a constraint satisfaction problem
           max_steps, the number of steps allowed before giving up

   current ← an initial complete assignment for csp
   for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen, conflicted variable from VARIABLES[csp]
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
   return failure
```

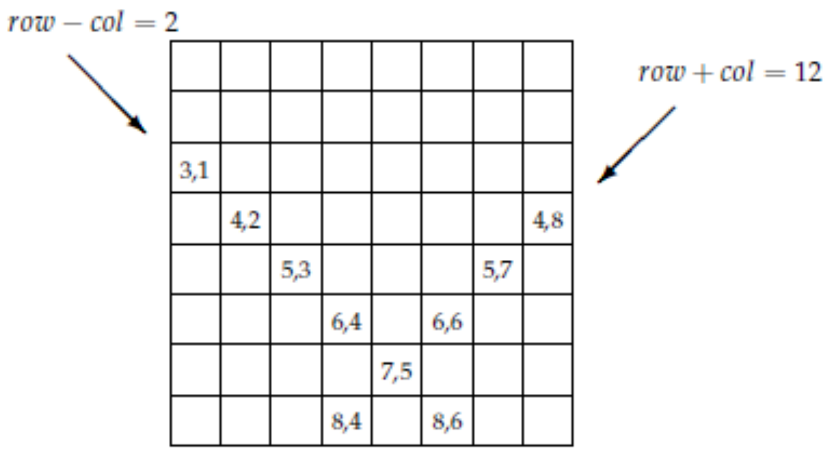
The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.



Efficiency

Diagonals can cause efficiency problems. The following notes can assist with counting conflicts, including diagonals.

Diagonals in the eight-queens problem



Observe that along a left diagonal (one that rises to the left), the *difference* between the row and column of the square stays constant. The square $(4,2)$ is in the same left diagonal as the square $(7,5)$, since $(4-2) = (7-5)$. The right diagonals are similar, except using the *sum*: the square $(4,8)$ is in the same right diagonal as the square $(7,5)$, since $(4+8) = (7+5)$.

Putting these ideas together, one can now state the general principle. A queen that is located on (row_1, col_1) can capture a queen that is located on (row_2, col_2) if and only if one of the following conditions holds:

- They are in the same row: $row_1 = row_2$.
- They are in the same column: $col_1 = col_2$.
- They are in the same left diagonal: $(row_1 - col_1) = (row_2 - col_2)$.
- They are in the same right diagonal: $(row_1 + col_1) = (row_2 + col_2)$.

Read “The Min-Conflicts Heuristic: Experimental and Theoretical Results” document from NASA posted on onQ for additional efficiency tips.