Assignment 1

Artificial Intelligence

CISC 352, Winter, 2021

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 nqueens 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: 5, Min Group Size: 3

N-Queens Problem.

Implement the iterative repair method using the minimum conflicts algorithm to solve the N-Queens problem. Collaboration on this project is restricted to your own group and the final submission will be via OnQ. The entire submission should be restricted to the provided file, **nqueens.py**.

Input.
Your program will read an input file called "nqueens.txt" which contains some number of 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 sample input file might be:
32
64
128

This indicates you need to produce solutions to, first, the 32-queens problem, then the 64-queens problem, and so on. You will be provided with a **run.py** file that reads from the file, and calls a method in **nqueens.py** – it is this "solve(..)" method that you must implement. You may create other supporting code, but make sure you keep it in this one file.

Output.
For each problem, the "solve(..)" method in **nqueens.py** should return a single list of integers. Each entry corresponds to the row of a queen and should be 1-based (i.e., the queen in the first row is at location 1, not location zero). The formatting of the output should be in lists such as:

Input:
4
5

Output file:
[2, 4, 1, 3]
[2, 5, 3, 1, 4]

Note that solutions are not unique. Further note that the local search 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. *Figuring out how to do this restart and repeated solution attempt is part of your task.*

Grading

The marking of this assignment will be done in two parts. The first part is based on the correctness and efficiency of your team's code. The second part is a written technical document.

The first section is out of 9 points. Three sets of problem instances of varying difficulty (small, medium, and large) will have a maximum of 4%, 3%, 2% respectively. Teams may lose marks by producing incorrect solutions and not meeting the efficiency requirements below. The following equation is how each stage will be calculated:

$$Percentage\ of\ Solved\ Boards \times Difficulty\ Multiplier$$

The difficulty test for each of the sections is as follows:

- **Small Difficulty (4%)**
  - Runtime >15 minutes on small n problems will result in a 0 for that problem
- **Medium Difficulty (3%)**
  - Runtime >25 minutes on medium n problems will result in a 0 for that problem
- **Large Difficulty (2%)**
  - Runtime >30 minutes on large n problems will result in a 0 for that problem

The second part is a grade on your technical document describing your algorithm. This is worth 3 points. The document will be marked on grammar, spelling, presentation quality, and the description of your algorithms. We should be able to know how your algorithms work from reading the document. There is no word count, however, please be concise and aim to write no more than two pages. 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 **with the contributions of each group member listed**. Convert the file to a PDF and named nqueens.pdf. Grading will be as follows:

- Clarity (is the algorithm explained in a way that is easily understood) **1 mark**
- Correctness (does the algorithm provide a valid answer) **1 mark**
- Overall quality (well-formatted, spelling, shows effort and thought etc.) **1 mark**

This assignment is **12%** 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.

Code Conduct: The code submitted will be assessed for accuracy and correctness. If there is any indication of inappropriate use (e.g., using other libraries/software to solve the problem), then marks will be deducted accordingly (up to 0/9 for the first section). We will also be making use of software to cross-reference all of the submissions (those from this semester and others we have on file). You are expected to undertake this assignment with the assumption that you work only within your assigned group for the duration.

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.