CS-203 – PROJECT 1 – 2023 FALL

GIUSEPPE TURINI – KETTERING UNIVERSITY

INTRODUCTION

N-Queens Problem

The 8-queens problem is a puzzle, originally proposed in 1848 by chess player Max Bezzel. The puzzle asks the player to place 8 queens on a traditional 8×8 chessboard so that no 2 queens are attacking one another. That is, the objective is to select 8 positions in an 8×8 square grid such that no 2 positions appear on the same row, column, or diagonal. [1]

The n-queens problem is a generalization of the 8-queens problem, in which (given a positive integer n) the goal is to place n queens on an n×n square grid such that no 2 queens are attacking one another. The problem has solutions for every n>3. [2]

The original 8-queens problem (n=8) has 92 distinct solutions; the number of solutions drops to 12 if one disregards solutions that are rotations or reflections of the original board. As an example, below (in Figure 1) you can see the only symmetrical solution (excluding rotation and reflection) to the 8-queens' problem.

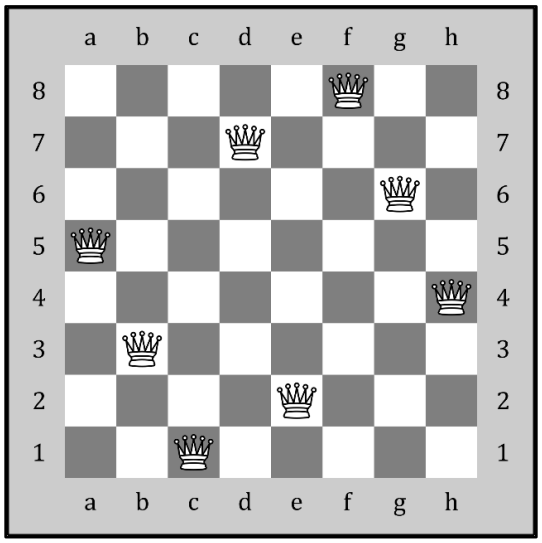

Figure 1. The only symmetrical solution (excluding rotations/reflections) to the 8-queens problem.

[1] See: Wikipedia – Eight Queens Puzzle.

[2] See: Google Optimization Tools – The N-Queens Problem.

Solution by Exhaustive Search or Brute Force

The n-queens problem can be easily solved by an exhaustive search algorithm or a brute force algorithm (these solutions will be discussed and partially implemented in class). Then optimizations can be made to reduce the running time of these simple algorithms (for example, by observing that any correct solution has exactly 1 queen in each row, and 1 queen in each column).

Solution by Iterative Repair

Another approach to solving the n-queens problem is the *"iterative repair"* algorithm. This program begins by placing 1 queen in each column, using random rows. If this placement yields a solution, the algorithm halts. If not, the algorithm perform column-swaps to reduce the *"attacks"* with the goal of reaching 0 attacks, when a solution is reached. [3]

THE PROJECT

Implement the Iterative Repair Algorithm

Write a Java program which, given in input n (n>3), finds a solution to the n-queens problem by iterative repair (see attachment), and prints the result to standard output. [4]

Analyze the Efficiency of your Algorithm

Analyze your algorithm performing both the theoretical analysis and the empirical analysis to evaluate the time efficiency. Each analysis should include:

- The analysis (description, method, formulas, efficiency class, data, etc.).
- Few visuals illustrating the analysis results (graphs, units, approximation function, etc.).
- Brief conclusions commenting the analysis results (comparison theoretical vs empirical).
- Brief comparison with simpler algorithms/solutions (exhaustive search, brute force, etc.).

Include these analyses in the technical report for this project (~3 pages in total).

[3] See: Wikipedia – Iterative Repair Algorithm.

[4] Sosic R, Gu J. *"A Polynomial Time Algorithm for the N-Queens Problem."* ACM SIGART Bulletin 1, no. 3, pp. 7-11. 1990.

SUBMISSION

Deadline and Procedure

Before Monday of 6th week, send an email to the instructor (gturini@kettering.edu):

- Send the email using your Kettering email account (subject: "CS-203: Project 1").
- The email must include in attachment your project (2 files, see below).
- Late submissions will be assessed a -5% penalty (-5 points) for each day of delay. [5]

Content

The content of your submission must be:

- A Java file ("SolverNQueens.java") with your code.
- A technical report ("CS-203-Project-1-Report.pdf") with your analysis.

[5] Note: the maximum penalty for a late submission is -30% (-30 points).

EVALUATION

This is the form used to grade this project.

PROJECT 1 – EVALUATION FORM

Implementation, Performance, and Analysis (70/100)

Iterative Repair Implementation (25/100)
Proper internal data representation, original algorithm structure, no deviations.

Iterative Repair Performance (15/100)
Proper internal data representation, attack data updated in constant time.

Theoretical Analysis (15/100)
Input size, best- and worst- cases, basic operation counts, efficiency classes.

Empirical Analysis (15/100)
Experiment setup, data collection, data visualization, proper result comparison.

Design, Style, and Submission (30/100)

OOP Design and Compilation Requirements (…/100)
Classes, fields, functions, variables properly designed, no unhandled exceptions.

Coding Style and Commenting (…/100)
File/class/function headers, variables comments, comment/code ratio, naming, indentation.

Submission Procedure and Delay (…/100)
Proper submission (email, subject, attachments), no delay.