Lab 3 – CSCI 112

**Due Date: Monday, Sep 14 at 5:00pm EST**

# Information

* This lab is intended to be completed **individually.**
* The files must be submitted with the exact file name provided in this file. If the file names do not match you will receive **zero** points for that file.
* Before you submit, make sure that your code runs. Any code which does not run without errors will receive **zero** points.
* Do not share your work with anyone other than Professor Khan or the TAs. You may discuss algorithms, approaches, ideas, but **NOT** exact code.
* If you submit work after the due datA second past the due date **WILL** be locked out from submission.

# Review

The N Queens problem is a famous logic puzzle. It is frequently limited to the 8 Queens problem, but can be generalized to any N by N board. The logic problem is: Given N queens and an N by N chessboard, how can you place the queens such that all queens are **safe**. Safe is defined by the queen's movement style in chess, where a queen can move horizontally, vertically, and diagonally. Additionally, given any N, **how many** solutions are there? You can read more about the N Queens problem [here](https://en.wikipedia.org/wiki/Eight_queens_puzzle).

In today's lab, the board is represented by a single dimensional list of length N. This represents our N queens, and is based on the assumption that there can be only one queen per column. Therefore, we only need to change the queens rows until we find a solution.

The easiest approach to solving this problem is to just **try all possible combinations of queen locations**. This is called a **brute force** method as it forces the solution. This starts with all queens in row 0. It then checks the board to determine if all queens are safe, checking all directions for each queen against **every other queen**. If the queens are safe, it is a viable solution. Otherwise, it moves the leftmost queen down and checks again, moves, etc. If the leftmost queen cannot move down (it would fall off the board because it is in the last row), then it resets the queen back to row 0 and moves the second queen down one. Think of it like a digital clock display, where each column can display a number between 0 and N-1. Once a column reaches the maximum number, it resets to zero and increases the column next to it.

A close up of a piano

Description automatically generated

# Assignment

**Task 1 – Analyzing Brute Force [5 points]**

Examine the current algorithm in the **nQueensBrute** function. This approach takes a **brute force method**, trying all possible combinations of queen positions in order until it finds a solution. Run **main()** in **nQueens.py** to see how long this approach takes given a board size, N. Note that sizes larger than 9 will take an **extremely** long time to finish and are not recommended.

Examine the **safeBoard**, **checkVertical**, and **checkDiagonal** functions. How much time would each of these take in big-O notation? Given an N by N board and trying to place N queens, how long in Big-O terms would this approach take to find **all** possible solutions? Write your answers in **analysis.txt**.

**Task 2 – Fancy Queens [10 points]**

The brute force approach can be greatly improved upon. Instead of placing all queens and then checking the entire board (all queens vs all other queens) for safety, a fancier version could place one queen and only move on to placing a new queen only if the current queen is safe. This way, if a placement would not work for a queen we don't waste time placing the other queens.

Fill out the **nQueensFancy** method so that it utilizes backtracking (fancy algorithm). You will need to create a new method called **safeQueen** to use instead of **safeBoard**. The **safeQueen** method will check one queen against all previous queens. Use the brute force function as a guideline as to how to utilize the **showTotal** parameter. The algorithm layout for **nQueensFancy** is as follows:

1. Initialize the board the same way **nQueensBrute** does
2. Start with the assumption that zero queens are safe, and keep track of a current queen you are trying to place safely
3. Infinitely loop:
   1. If the current queen is safe:
      1. If the current queen is the last queen:
         1. If calculating a total, increase the total by one and move the current queen down
         2. Otherwise, return the board as we've found a solution
      2. Otherwise, move to the next queen
   2. Otherwise:
      1. Try to move the current queen down
   3. While the current queen is off the edge of the board:
      1. If the current queen is the first queen, we've run out of possibilities and return the total
      2. Otherwise:
         1. Reset the current queen to the first row
         2. Move back one queen
         3. Move the new queen down one row

You can test your accuracy with **showBoard**. Add your **nQueensFancy** to the call to **compare** in **main** and run **main** to see how the fancy approach compares in runtimes.

A close up of a piano keyboard

Description automatically generated

# What To Turn In

Create a zip file named **Lab3\_<your W&L ID>.zip**.Inside this zip archive should be **nQueens.py, tools.py** and **analysis.txt**. No other files are needed. Make sure your files have the names (including case), and extensions.