Lab 04 Specification – Exhaustive Search – Eight Queen
Due (via your git repo) no later than 8 a.m., Thursday, 18th April 2019.
50 points
[An individual two-week lab assignment. This is a MAKE-UP lab.]

Lab Goals

- Implement an exhaustive search algorithm to solve the Eight Queens problem.
- Upgrade your algorithm to handle more generic cases.
- Answer some more questions relating to course content.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`
- Often times, the lab specification is purposefully developed to include open ended tasks and not exposing too low level details. The intention is to trigger you towards further refining your thinking skills. Thinking is more important to solve any algorithmic problem provided. Thinking through, may lead you to ask further clarifying questions, which is what my expectation is. So, there will be one or more open ended parts added to the lab. Attending labs regularly and steering discussions with the Professor, TA's and your classmates is going to be the success mantra for doing a great job in the labs. Waiting till the last minute, will restrict a student from both learning the content discussed and to have a negative effect to the effectiveness of the lab work produced.

Learning Assignment

If you have not done so already, please read all of the relevant "GitHub Guides", available at https://guides.github.com/, that explain how to use many of the features that GitHub provides. In particular, please make sure that you have read guides such as "Mastering Markdown" and "Documenting Your Projects on GitHub"; each of them will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- Sedgwick chapter 02 [as recommended in the reading assignment section of the class slides for any recursive sorting algorithm such as Quick or Merge Sort. Understading how recursion works is a key component to do well in this lab.]

Assignment Details

One topic that we likely will not have time to discuss in this class is the idea of Exhaustive Search. In exhaustive search, we find a solution to a problem by considering (possibly) all potential solutions, and selecting the correct solution from the list of all potential solutions. The run time of an exhaustive search problem is bounded by the number of possible solutions to the problem. That means that if the number of potential solutions is exponential, then the run time will also be exponential. Exhaustive search is often also commonly called Brute Force.

Exhaustive search algorithms can often be effectively implemented using recursion. Each recursive call progresses one node down the recursive tree. When a recursive call terminates, control reverts back to the calling parent, which then resumes execution. This allows us to easily implement Backtracking. In backtracking, an algorithm proceeds forward towards a solution until it becomes apparent that no solution can be achieved along the current path. At that point, we undo the current solution attempt (backtrack) to a point where we can again proceed forward.

In this lab, we will use exhaustive search to find solutions to the Eight Queens problem (defined in `Part One`). After discovering an algorithm to find one solution, the algorithm will be modified to display all solutions.

The lab can be implemented in any programming language. Each individual can make their own decision to choose an appropriate programming language based on the individual's strength, weakness, and personal choice. But, since we had used primarily Java as the programming language to implement our in-class discussion points and ideas, there are two bash files provided along within the repository to compile and run your program in a Java environment. It is required for you to provide a read me file that containts detailed instructions on how to compile and run your program.

Part One: Implementing Basic Eight Queens

The Queen is the most powerful piece on a chessboard. Queens have the ability to move horizontally in a row, vertically in a column, or diagonally across both a row and column. Queens can also move any number of spaces, ranging from one space outward to the edge of the board. This means that it is possible for a single Queen to "attack" as many as 27 of the 64 squares on a standard 8x8 chessboard.

The goal of the Eight Queens problem is to identify a way to position 8 different Queens on a single chessboard such that no Queen can directly attack another Queen. A simpler way to think about this problem is that a Queen should be positioned in each row, a Queen should be positioned in each column, and no Queen should intersect the diagonal movement of another Queen. It turns out that there are 92 different ways to position 8 nonconflicting Queens on an 8x8 chessboard. The trick is identifying one of these 92 solutions.

Here are some bad ways to solve Eight Queens: A naïve brute force algorithm will blindly place 8 Queens on a chessboard, then check to see if it is a valid solution. This blind placement of 8 Queens will consider all 64^8 possible blind placements of 8 Queens, yielding 281,474,976,710,656 possible solutions. Clearly this is inefficient. (Wolfram Alpha notes that this is 14 times the number of red blood cells in the average human body.) We can greatly reduce this number of potential solutions by filtering all solutions that place multiple Queens on the same square of the board. This results in $\binom{64}{8}$, or 4,426,165,368 possible solutions. Better, but still a large search space. We can limit this solution space even further by including the restriction of a single Queen on every row. Now we have reduced our solution space to $8^8 = 16,777,216$ possible solutions.

This is where we will begin our recursive, backtracking exhaustive search. We can design a recursive function `placeQueen()` which will cycle across a row, trying to place a Queen in each column. If the placement is legal, then a recursive call is made to the next row. If the placement is not legal, then we iterate to the next column and attempt to place the Queen there. If we make it to the end of the row without finding a valid Queen placement, we backtrack out of the current recursive call and into the previous, to try to place the Queen in a different column in the previous row. If we find a valid location for Queen placement in the last row, then we have found a solution. A framework for the recursive function is below:

```
placeQueen(int row) {
    for (int col = 0; col < 8; col++) {
        if (isLegalPlacement(board, row, col) {
            addQueen(board, row, col);
            if (row == 7) {
                printSolutionAndExit();
            } else {
                placeQueen(row+1);
            } //if-else
```

```
        } //if
    } //for
} //placeQueen
```

In this section, you will implement this recursive algorithm, and display the first solution that the algorithm generates.

Part Two: Implementing Advanced Eight Queens

From here, you will make two small tweaks to your Eight Queens algorithm.

1. Rather than printing a single solution to the Eight Queens problem, you should modify your program to print all 92 of them. That is, rather than exiting after a solution is generated, you should remove the most recently placed Queen, and continue through the recursive calls until a second solution is found. Print that second solution, remove the most recently placed Queen, and continue the recursive backtracking process. A framework for this recursive function could look like the following:

   ```
   placeQueen(int row) {
       for (int col = 0; col < 8; col++) {
           if (isLegalPlacement(board, row, col) {
               addQueen(board, row, col);
               if (row == 7) {
                   printSolution();
               } else {
                   placeQueen(row+1);
               } //if-else
               removeQueen(board, row, col);
           } //if
       } //for
   } //placeQueen
   ```

2. Modify your Eight Queens code to solve the N-Queens problem – print one (or all) solutions for placing N Queens on an NxN-sized chessboard. This should hopefully be a simple code modification as well – simply alter the number of recursive steps and number of columns iterated across. You can verify that your code is working correctly by creating a counter recording the number of solutions found. The table at the top of the next page lists the number of solutions for various board sizes. A framework for this recursive function could look like the following:

   ```
   placeQueen(int row) {
       for (int col = 0; col < numColumns; col++) {
           if (isLegalPlacement(board, row, col) {
               addQueen(board, row, col);
               if (row == numRows-1) {
                   printSolution();
               } else {
                   placeQueen(row+1);
               } //if-else
               removeQueen(board, row, col);
           } //if
       } //for
   } //placeQueen
   ```

N	# Solutions
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200

Part Three: While You Have Some Downtime

Here are some questions to test your knowledge of class content:

1. Trace the process of inserting the keys EIGHTQUEENS into a binary search tree. Each key is associated with the value corresponding to the index of the letter in the string (ALPHA). Show the final tree after inserting all the characters into the tree.

Submission Details

For this assignment, please submit the following to your GiHub repository by using the link shared to you by the course instructor:

1. Commented source code from the “PartOne” program under the src sub-directory.
2. Commented source code from the “PartTwo” program under the src sub-directory.
3. A readme file explaining how to compile and run your program. In addition, include a detailed self reflection of the lab exercise in the readme file. The self reflection may include answers to questions such as:
   - ”What challenges did you face in this lab?”
   - ”What did you like in this lab?”
   - ”What did you not like in this lab?”
4. A PDF file that contains your answers to Part 03 of this lab. You may draw the tree diagram in the paper and integrate a picture of it in the PDF file.
5. It is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine, unless otherwise cited.” in all your deliverables such as source code and other textual documents.

Grading Rubric

1. The lab is worth 50 points. It is expected that the submission has the complete solution to the problem above. The breakdown for this assignments points is :
   - Part one: 25 points
   - Part two: 10 points
   - Part three: 10 points

- Documentation (README file) : 5 points

2. There will be full points awarded for the lab, if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.

3. Failure to upload the lab assignment code to your git repo, will lead you to receive ”0” points given for the lab submission. In this case, there is no solid base to grade the work.

4. There will be minimal partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in github. In this way, an updated version of team’s submission will be used for grading. If the team did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the team had not submitted any code, then in this case, there is 0 points given to the team automatically for the lab work. All team members will get the same grade for the actual lab part except the points allocated to the group meetings.

5. If you needed any clarification on your lab grade, talk to the instructor. The lab grade may be changed if deemed appropriate.