Lab 05 Specification – Exhaustive Search – Eight Queen
Due Friday, 30th March 2018 1PM
Please refer to the syllabus for information about late submission policy.
Total - 50 points.

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.
- This is a 4 weeks lab, to accommodate Spring break and Midterm exam which accounts for 2 weeks out of the 4 weeks time period.
- Due to the additional time provided to the lab, this lab is going to be an individual lab. You can collaborate with your team mates, but only on a very high level. You cannot exchange code, and your work should be unique and completely implemented by you.

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.

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:

```java
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:

```java
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 \(N \times N\)-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:

```java
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
```

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 symbol table. Each key is associated with the value corresponding to the index of the letter in the string. List the final set of Key-Value pairs after all letters are inserted.
2. Which symbol table implementation (Sequential or Binary Search) would you choose for an application that runs $10^3$ put() operations and $10^6$ get() operations?
Submission Details

1. You are required to submit this lab by sending an email with a zipped version of your cmpsc250-lab05-YOURFirstInitialLastName folder.

2. Subject of your email should say "CMPSC250: YOUR_LAST_NAME Lab 05 Submission". Here YOUR_LAST_NAME needs to be replaced with your last name.

3. Send the email to amohan@allegheny.edu

4. You should add the following statement in the body of your email
   By doing this submission, I understand that I am subject to the Honor Code policy.
   Lab submitted by: X (Here X is your full name.)

5. Your source code for EightQueens.java (or whatever you decide to name your file).

6. To show that you have completed Part One, show an output for the first solution to Eight Queens.

7. To show that you have completed Part Two, show the first five solutions for Eight Queens, and show the first solution for both Six-Queens and Twelve-Queens.

8. The answers to the questions in Part Three as PDF file inside your folder.

9. Provide README file inside your code, that will serve as your documentation and it should give details on how to run your program, give a short description of your algorithms and their worst case complexity and any additional sources you have used.

Points Distribution

The breakdown for this assignments points is :

- Part one: 25 points
- Part two: 10 points
- Part three: 10 points
- Documentation (README file) : 5 points

Note

- Although this is not due for four weeks, I expect you to be in the lab on March 09th to continue working on your lab and to ask me any questions or clarifications you may have. Other lab sessions during this period are, March 16th (Exam), March 23rd (no classes - spring break), march 30th (due date).