**CS 1181**

**Programming Assignment 5**

PURPOSE: The purpose of this lab is to give you an opportunity to make use of a stack in the implementation of a backtracking algorithm.

Description: This problem is generally referred to as the N Queens problem. The gist of the problem is to create an algorithm which can position N queens on a chess board so none of the queens can attack any other queen. For those of you who may not be familiar with chess, the queen can move horizontally, vertically, and diagonally. In the generalized N Queens problem, N represents the number of rows and columns of the game board and the number of queens that must be placed on the board in safe positions.

This particular problem is an interesting problem to solve using a recursive approach, but since we have not studied recursion yet, we will **NOT** be using recursion in any way to solve this problem.

The basic strategy that we will employ to solve this problem is to use backtracking. In this particular case, what backtracking means is that we will make a move that is safe for a queen at the time we make the move, but later on, we may discover that we are stuck and there is no safe move available for the next queen we are trying to place on the board, so we must backtrack. That means we will go back to the previous queen’s move, undo that move, and continue searching for another safe place to put that queen. We will use the stack to remember where we have placed queens on the board and when we need to back up, the queen at the top of the stack will be popped and we will use that queen’s position to resume the search for the next safe location.

Here are some observations which can help in understanding how to approach the problem and how to organize data in your program to keep track of the state of your search.

* There are N rows. When a queen is placed into a row, that row cannot be used by other queens.
* There are N columns. When a queen is placed into a column, that column cannot be used by other queens.
* There are 2 sets of diagonals to be concerned with, diagonals running upper left to lower right and diagonals running upper right to lower left. There is a simple mathematical relationship between N and the number of diagonals in each direction. When a queen is placed onto the board, one left to right diagonal is no longer safe for other queens, and one right to left diagonal is no longer safe for other queens.

Our approach to placing the queens on the board will be to use nested loops. As an example the outer loop will be in control of which column we are currently searching for a spot to place a queen. So, the first queen we place will be in column 0, and when the outer loop has completed we will have placed N queens on the board, one in each column.

The inner loop will iterate through the rows, from 0 to N-1 until it finds a row **R** such that row **R** is not taken by another queen, **AND** the 2 diagonals which intersect row **R** and the current column are not taken by another queen. **IF** we find such a safe location, we push the current row, column value onto a stack. Note: we can use the Java Point class to store the row and column numbers in the x, y values of a Point object. By placing a queen’s location information into the stack, we have affected the state of the board. We must update the state information to reflect that a row is no longer safe for other queens. We also must update the state of the two diagonals which intersect with the queen’s row, column position to indicate that those diagonals are no longer available to other queens to be placed later. After updating the state information, we can break out of the inner loop and continue on to the next column in the outer loop.

The backtracking aspect of this exercise appears when the inner loop fails to find a spot to place a queen in the current column. Inside the inner loop, we need to recognize that we have checked position (row N-1, current column) and found that it is not safe. Therefore there is no safe place for a queen in this column. Since we are stuck, we need to back up and try a different placement for the previously placed queen. To backtrack, we need to undo the actions taken when we placed the last queen onto the stack. That means pop the last queen off the stack, mark the row and diagonals associated with that queen’s position as available again, and reset the row and column loop control variables back to the point where they were when this queen was pushed onto the stack (ie row = poppedPoint.x, column = poppedPoint.y). After backtracking the row loop will continue to the next row to check if that row, column position is safe.

Once the loops complete their work, the stack will contain the points (x,y) where the queens are located. At this point, you need to find a way to output the state of the game board in order to show that all your queens are positioned safely. Here is an example output for 9 queens.

X X X X Q X X X X

X X X X X X Q X X

X X X X X X X X Q

X X X Q X X X X X

X Q X X X X X X X

X X X X X X X Q X

X X X X X Q X X X

X X Q X X X X X X

Q X X X X X X X X

Think very carefully about how you want to organize the data which captures the state of the game board (rows, diagonals). You do not need an actual representation of the game board (ie. no 2 dimensional array). Also think carefully about how to mathematically capture the relationship between a row, column position and which diagonals intersect that position. Part of creating efficient programs is designing clever data representations which can be accessed in very efficient ways from known information.

Also note, the value of N should be able to be easily changed. You can ask the user what value of N they want to use and then control everything based on the user’s input. You might want to limit the user’s input to something under 50. The minimum value of N must be 4.

**Save your Netbeans project by creating a ZIP file. Copy all your source code into a text document. Upload your ZIP file and your text document to the Pilot drop box before the lab due date.**

**No static variables are allowed. The use of static variables will result in a 20% deduction.**

**Rubric (100 pts)**

**Your program will receive a 0 if it does not compile OR if it uses recursion in any way.**

* **User can enter the value of N and the program adjusts to the user’s input (10)**
* **Program structure adheres to basic nested loop (15)**
* **Use of stack consistent with lab description (10)**
* **Backtracking to adjust row and diagonal state works (10)**
* **Structure to keep track of row state used properly (10)**
* **Structures to keep track of diagonal state used properly (10)**
* **Output of the final position of N queens on the board clearly shows whether program worked (10)**
* **Program correctly finds positions to place 9 queens on the board (15)**
* **Commenting and Code Formatting (10 pts)**
  + **All methods have method level Javadoc comments consistent with the style guidelines**
  + **All code adheres to style guidelines**