# Data structures and algorithms, assignment two:

# The eight-queen puzzle is the action of placing eight queens on a chessboard so that no two queens are attacking each other. However; this assignment task differs from the standard eight queen puzzle as in this assignment I have to create a programme which allows the user to place the first queen on an eight by eight chessboard and print out all the possible solutions of positioning the queens on the board. Meaning I must consider all possible outcomes in which the other seven queens can be placed on the board without any of the eight queens attacking each other. There are many approaches to this problem one of them being brute force; which consists of checking all positions on the grid & whether an occurrence of the pattern starts there. This means that in a brute force algorithm the program will go through all of the possible solutions extensively, without taking in to consideration the final solution. Brute force algorithms have a complexity of O(mn), meaning the program is doing a constant amount of work for each value of the matrix.

# The method in which I used to carry out this eight-queen problem is partially similar to brute

force algorithms I carried out solving the eight-queen problem using backtracking. The

backtracking algorithm is a technique for solving a problem recursively; by trying to build a

solution which works incrementally, going through each different position. Removing each solution which fails to meet the requirement of the eight-queens not attacking each other. If they find a solution which does not satisfy the requirements of the individual queen not being attacked in the vertical, horizontal, and diagonal direction we simply remove it which is the method of backtracking and then attempt to try a different position. If we simply focus on the actual recursive function of backtracking in terms of branching possibility of the queen at each position on the eight by eight grid the algorithm will have an exponential complexity of O(2^n). I feel backtracking was the most effective way in which possible to approach this problem, as backtracking allows me to work through each protentional position of the chessboard and see whether this position is a possibility for positioning one of the eight queens. The backtracking algorithm decides between multiple different alternatives to the next component of the solution and it simply tries all the possible different options recursively. Hence the name backtracking, the algorithm backtracks once it has not found a value which will not satisfy the constraints of the program.

Backtracking is seen commonly within many computing practices the most popular one

being, depth first search which uses the concept of backtracking. As in depth first search, we

explore all the possible paths of each given node recursively until we reach the solution we aim for. There are two possible outcomes when implementing backtracking within my n queen program, one possible solution is that we have found all

possible solutions in which we simply exit and print these solutions to the terminal. The other possible outcome to the program is that we did not find the end state which we were aiming for, in this protentional scenario we simply backtrack to the previous checkpoint -working solution- and then attempt to place the queen in a different position.

The hardest part of this assignment was the queen’s freedom of movement which makes the n queen problem particularly difficult. As due to the freedom of the chess piece, we must check every potential position in which an attacking queen could be positioned. This means we must recursively check each position which is on the same vertical, horizontal, and diagonal line of the new queen which is being positioned. I feel backtracking was the most efficient way in which I could have approached the problem to travel through the positions on the chessboard, until finding a suitable solution.

**Pseudo code:**

1. If it’s the users selected row increment queens and go to the next row
2. If all queens are placed and satisfy the constraints of the program return true.
3. If the placement of the queen does not satisfy constraint of the program remove the queen and backtrack to previous step and attempt to try other rows.
4. If all rows have been attempted and this does not form a solution which satisfies constraints return false to trigger backtracking.

My eight-queen program consists of three classes, chessboard.java, main.java & BIO.java

**References**

**BIO.java-** [**http://www.cem.brighton.ac.uk/staff/mas/mas/courses/ci101\_gen/cw1/BIO\_java.html**](http://www.cem.brighton.ac.uk/staff/mas/mas/courses/ci101_gen/cw1/BIO_java.html)