CS 310
Assignment 424

Brandon Ingli

Problem 3b. Write a brief paper analyzing the efficiency of your test function. Analyze my function as it currently exists, analyze your test function after you improve it, and explain the improvement.

Answer: Let's start by analyzing the original test function. The input size is the number of queens to place on the board, n, with a resulting board size of $n \times n$. Where the queens are placed does not affect this function, and it cannot end early. The function features four nested for loops whose headers each run n times, and a constant number of comparisons, assignments, and other local work within the nested loops. Therefore, this function can be classified as

$$T(n) \in \Theta(n^4)$$

Now, let's analyze my test function. The input size is once again the number of queens to place on the board, n, resulting in a board size of $n \times n$. The placement of queens does have an effect on this function, and it can end early. There are two while loops nested, each running a maximum of n times. Inside the nest is a constant number of assignments and comparisons. The best case for this function is if queens are placed at positions (0, 0) and (0, 1). The inner loop body will run a constant number of operations twice before the boolean flag is raised and the function terminates. Therefore, the best case efficiency can be expressed as

$$T(n) \in \Omega(1)$$

The worst case for this function is finding a valid board. Each loop header will run n times, and the body within the nested loops will run a constant number of operations each iteration. Therefore, the worst case efficiency of the function can be expressed as

$$T(n) \in O(n^2)$$

This dramatic improvement can be attributed to the fact that, while the old function iterates through the entire matrix for each cell of the matrix, my function only iterates through the entire matrix once. This is achieved by utilizing four vectors of booleans to remember the rows, columns, and diagonals where queens have been seen before. The old function would not remember where queens were and would instead search for them each time a new queen is found. Additionally, my function employs while loops, which can terminate as soon as a violating condition is found, while the old function employs for loops, which must run through every iteration, even when a violating condition is found. This means that needless calculations are not performed in my function, increasing its efficiency. As this function is run for every node visited in the search tree, every little bit of extra efficiency that can be squeezed out of the test function makes a measurable impact in the overall efficiency of the backtracking program.