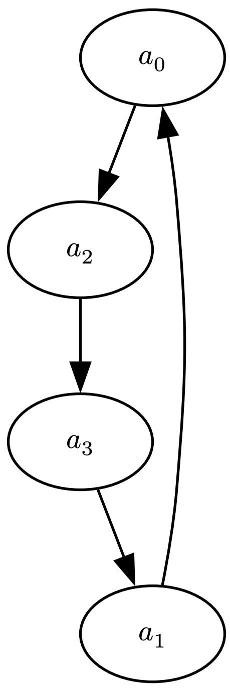
The N-Queens Problem as a Digraph

The N-Queens problem is a rather simple one: if you have a $N \times N$ chess board (for fixed $N \in \mathbb{N}$), can you place N queens such that none of them are checking each other, or in other words, so that none of them are able to take each other. A more detailed description is given at https://en.wikipedia.org/wiki/N_queens. We can visualise this as an $N \times N$ matrix, where 1 represents a queen being in a specific position, and 0 represents an empty tile. For example, the matrix

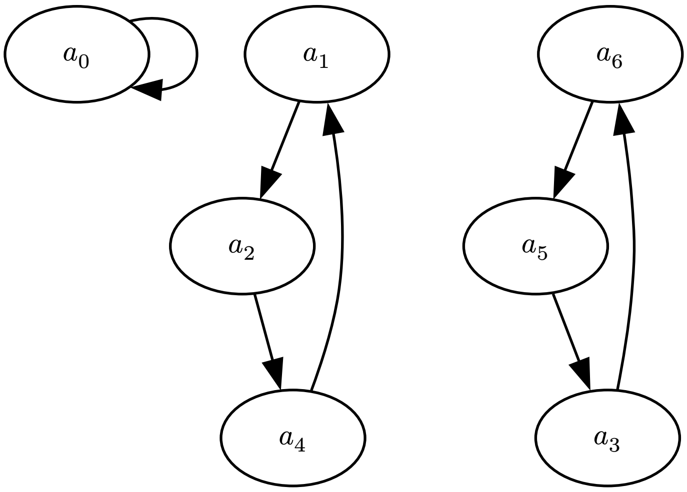
$$M := \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

is a valid solution, as none of the 1s can "attack" each other on a diagonal, horizontal, or vertical line (the queen's valid moves). Notice that this could form a digraph, where the top of the matrix is labelled $a_0, a_1, ..., a_N$, and the side $b_0, b_1, ..., b_N$, and an edge from a_n to b_m (with $n, m \in \{0, 1, ..., N\}$) existing iff $M_{m,n} = 1$ (where M is the matrix in question). Thus, our matrix above would form the digraph



Notice the cycle in the digraph. It is also important to note that there may exist multiple solutions to a given N. When we checked the first 8 solutions computationally, we found that *all* of the corresponding digraphs had similar cycles (the code is available in the Github repository https://github.com/AowynB/NQueens, along with the source for this document, and a copy of the PDF). Some of them – for example, at $N = 7$ – had multiple small cycles, the matrix and digraph for which is shown below.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$



We seek to show that this pattern of cyclic digraphs holds in general iff the matrix exists. Note there are cases where no solution is possible, namely $N = 2$ and $N = 3$. First, we need to understand what the digraph is representing about the board. [Still needs to be filled in.]

The Algorithm

[I'll describe the algorithm in more detail later.]

Much of guts our algorithm was inspired by a wonderful paper by Richards [1]. Please note that this is a rough outline of the algorithm; later, we'll describe a more efficient approach. When computing valid solutions to N-Queens, we start with an empty board, and then place down all possible Queen positions on the first row across four different matrices. This gives us N matrices that we're currently working on. Now, we repeat the process, starting on the next row, for each of these matrices, creating N copies of it, and then placing a Queen in each spot on the row. We now have $N^2 + N$ matrices. However, we can cut this down substantially by pruning all solutions that form an invalid solution, that is, the Queens can take each other. Due to this pruning, we'll have strictly less than $N^2 + N$ matrices. We repeat this process until we have explored all possible routes through this tree. Noting that the N-th level of our tree has N^N vertices, it follows that to reach the N-th level, it'll take $\Sigma_{i=0}^{N} N^i$ space. Thus, we have $\mathrm{O}\left(\Sigma_{i=0}^{N} N^i\right)$ as an upper bound to our space complexity (we will improve on this later, once the algorithm is shown in all its glory).

Now, assume that each new level of the tree to take $\Theta(1)$ time to compute (I.E., we have unlimited cores, and we are using N of them to compute the N-th level). Then, to reach the N-th level, it will take us $\Sigma_{i=0}^{N} 1$ iterations, which gives us an asymptotic time complexity of $\Theta(N)$.

In the real world, the time complexity will be worse, as we do not have this idealised computer, and thus must put an upper bound the number of simultaneous processes. However, the space complexity will be better, since we can apply a set of "clever tricks" to cut down on the number of possible things we need to check, though it will still be bounded by the space and time usage.

Bibliography

[1] M. Richards, "Backtracking Algorithms in MCPL using Bit Patterns and Recursion." 2009.