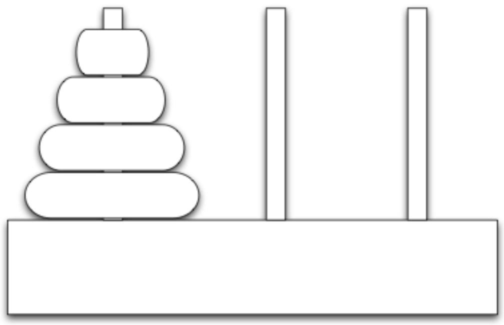
CS 188 Summer 2019 Section 1: Search

1 Towers of Hanoi



The Towers of Hanoi is a famous problem for studying recursion in computer science and recurrence equations in discrete mathematics. We start with N discs of varying sizes on a peg (stacked in order according to size), and two empty pegs. We are allowed to move a disc from one peg to another, but we are never allowed to move a larger disc on top of a smaller disc. The goal is to move all the discs to the rightmost peg (see figure).

In this problem, we will formulate the Towers of Hanoi as a search problem.

(a) Propose a state representation for the problem

One possible state representation would be to store three lists, corresponding to which discs are on which peg. If we assume that the N discs are numbered in order of increasing size $1, ..., n$, then we can represent each peg as an ordered list of integers corresponding to which discs are on that peg.

(b) What is the start state? $([1, ..., n], [], [])$

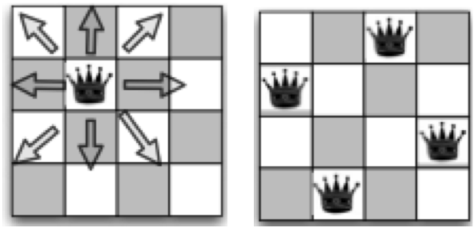
(c) From a given state, what actions are legal?

We can pop the first integer from any list (i.e., peg) and push it onto the front of another list (peg), so long as it is smaller than the integer currently at the front of the list being pushed to (i.e., peg being moved to).

(d) What is the goal test? Is the state the same as $([], [], [1, ..., n])$?

2 n-Queens

Max Friedrich William Bezzel invented the eight queens puzzle in 1848: place 8 queens on an 8×8 chess board such that none of them can capture any other. The problem, and the generalized version with n queens on an $n \times n$ chess board, has been studied extensively (a Google Scholar search turns up over 3500 papers on the subject).



Queens can move any number of squares along rows, columns, and diagonals (left); An example solution to the 4-queens problem (right).

a) Formulate n-queens as a search problem. Have each search state be a board, where each square on the board may or may not contain a queen. To get started, we'll allow boards in our state-space to have any configuration of queens (including boards with more or less than n queens, or queens that are able to capture each other).

Start State: An empty board

Goal Test: Returns True iff n queens are on the board such that no two can attack each other

Successor Function: Return all boards with one more queen placed anywhere. Another possibility (see part d) - place queens left to right (i.e. in the first column, then the second column, etc.)

b) How large is the state-space in this formulation? There are n^2 squares, each of which may or may not contain a queen. Therefore there are 2^{n^2} possible states, or 1.8×10^{19} for 8-queens.

c) One way to limit the size of your state space is to limit what your successor function returns. Reformulate your successor function to reduce the effective state-space size. The successor function is limited to return legal boards. Then, the goal test need only check if the board has n queens.

d) Give a more efficient state space representation. How many states are in this new state space? A more effective representation is to have a fixed ordering of queens, such that the queen in the first column is placed first, the queen in the second column is placed second, etc. The representation could be a n-length vector, in which each entry takes a value from 1 to n, or "null". The i-th entry in this vector then represents the row that the queen in column i is in. A "null" entry means that the queen has not been placed.

Since each of the n entries in the vector can take on $n+1$ values, the state space size is $(n+1)^n \approx 4.3 \times 10^7$ for $n = 8$.

To further limit the state space, we can require that queens are placed on the board in order (in this case, left to right). Now we know that if k queens have been placed on the board, the first k entries in the state space are non-null and the last $n-k$ entries are null. This creates a total state space size of $\sum_{k=0}^{n} n^k = \frac{n^{n+1}-1}{n-1} \approx 1.9 \times 10^7$ for $n = 8$.

Finally, by combining this idea with the successor function in part (c), we can further limit the *effective* state size.