Solution for Pyramid

The Pyramid problem was a generalization of the well-known Towers of Hanoi problem, and in this solution we will use the traditional terminology – the slices we move will be called discs, and the three locations are pegs.

Note that we get the original problem for example in the case when all weights are equal to 1, and strengths are 0, 1, ..., N-1.

The original problem can be solved recursively: To transfer the entire tower from peg 1 to peg 3, we have to move the largest disc at some point. This is only possible if nothing is on top of it, and if peg 3 is free. This means that to transfer all N discs from peg 1 to peg 3, we first have to (1) transfer N-1 discs to peg 2, then (2) the largest disc from 1 to 3, and finally (3) transfer the N-1 discs from peg 2 to peg 3. Steps (1) and (3) are done recursively, that is, by applying the same algorithm. It can easily be seen that we need 2^N-1 moves to transfer the entire tower.

In our more general problem, this is always an upper bound on the optimal solution – as in the original tower each disc can hold all discs above it, we can always number them 1 to N in the order in which they appear in the input, and apply the original algorithm.

However, in many cases such a solution can be far from the optimum. For example, in a test case where the strength of each disc exceeds the sum of the others' weights, just N moves are enough to transfer the entire tower.

In general, the problem is hard, and no optimal solution in polynomial time is known to the authors. We will now explain several possible strategies how to approach this problem.

The total number of reachable states can be bounded from above by (N+2)!/2. The reasoning behind this is that each state can be written in the form “contents of peg1 | peg2 | peg3”, which is essentially a permutation of N numbers and two separator symbols. This means that roughly up to N=9 or even N=10 a complete state space search is possible, and one can best implement it using breadth first search.

For larger inputs such an approach would only work for inputs that resemble the original problem. Note that in the original Towers of Hanoi problem there are only 3^N valid configurations, as the configuration is uniquely specified by the peg numbers of the discs, in order. This means that for similar inputs the number of reachable states will actually be much lower than (N+2)!/2.

Further speedup can be achieved using good implementation techniques, such as packing the state into a few integers – with the complete state space search memory consumption starts to be an issue pretty soon.

For larger inputs, a viable alternative to the breadth first search is the A* search algorithm, using a suitable heuristic function h(S) that gives us a lower bound on the number of steps we need to make from state S to reach the goal. One simple function is to try all possibilities which of the incorrectly placed discs will be the next one on peg 3, and then counting the moves supposing that the discs have infinite strengths. Such a heuristic function helps to cut the search space significantly, while keeping the solution correct – as soon as we reach the goal, we can be sure that the number of moves made is optimal.

For test cases that even the A* search is not able to solve in a few minutes, one has to use some heuristic approach. Some ideas are listed below:

- Try to find clusters of interchangable discs and treat them as one new disc.
  Afterwards, if you managed to reduce the number of discs to a reasonably low number, apply the optimal algorithm.
- Apply the optimal algorithm to the top K discs, for some K. From now on, consider them as a single disc and move them all at once, using the optimal way you found. Recursively handle the remaining problem. Try various values of K and pick the best one.
- Use dynamic programming to find the best way to split the input into smaller clusters for the above approach.
- Just use dynamic programming to find the optimal way to split the input into chunks such that you can keep each chunk together and use the naïve algorithm to move them.
- Each time peg 3 is “free” (only contains discs that will not move any more) and you can move a disc that can hold all others, it is optimal to be greedy and move it to peg 3.