Code Jam 2009 - Round 3

Analysis: EZ-Sokoban

This is a state space search problem: given a set of states (positions on the board), an initial state and a final state, and rules for state transformations, find a sequence of moves that transforms the initial state to the final state. In our case, the problem asks for the length of the shortest such sequence of moves.

Conceptually, we can represent such a state space as a graph. The nodes of the graph are the possible positions, and the edges are the allowed moves. The problem then becomes: find the shortest path in the graph. The standard algorithm to solve this graph problem is breadth-first search.

Let's estimate the number of nodes in the graph. Assuming the maximum number of boxes (5), first estimate the number of positions where all the boxes are connected. 5 connected boxes form a pentomino. There are 63 different pentominoes, counting all rotations and reflections. Each of these can be positioned at no more than 12*12 different positions. Hence we get an upper bound of 63*12*12 = 9072 connected positions. It is a little more difficult to estimate the "dangerous" positions accurately, but we can see that from each connected position there are not too many moves, so we can guess that the total is not going to be too large for our computer to handle.

One approach would be to first generate the graph with all the edges explicitly, and then run the breadth-first-search on it. Another is to not store the graph at all, but compute the possible moves (edges) from a given position as we go, and only store the set of visited positions in a data structure.

Some details to work out are:

- How to represent positions. A simple list of box coordinates can work. One can also use a whole-board bit-mask.
- How to look up positions. We need this to see if a position has already been visited, or to avoid constructing the same node in the graph multiple times. We can use some kind of a dictionary data structure - a hash table or a binary search tree.
- How to generate moves. Just try moving all the boxes in every direction, if the space in front and in the back of the box is empty. We also have to make sure that we don't move from a dangerous position to another dangerous position.
- How to check whether a position is dangerous. To do this, we need to check if our 1 to 5 boxes are all connected. This can be represented as another, small graph problem (graph connectivity). We run another breadth-first-search on the little graph, where the nodes are the boxes and the edges indicate whether two boxes touch. Another approach would be to pre-generate all polyominoes of sizes up to 5, store them in a hash table, and then look up the shape appearing in a given position.