CS 325 Group Project 3

Robert Detjens, Srikar Valluri, Felix Brucker

Scenario

The infamous Mongolian puzzle-warrior Vidrach Itky Leda invented the following puzzle in the year 1437. The puzzle consists of an n × n grid of squares, where each square is labeled with a positive integer, and two tokens, one red and the other blue. The tokens always lie on distinct squares of the grid. The tokens start in the top left and bottom right corners of the grid; the goal of the puzzle is to swap the tokens.

In a single turn, you may move either token up, right, down, or left by a distance determined by the other token. For example, if the red token is on a square labeled 3, then you may move the blue token 3 steps up, 3 steps left, 3 steps right, or 3 steps down. However, you may not move either token off the grid, and at the end of a move the two tokens cannot lie on the same square.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given Vidrach Itky Leda puzzle, or correctly reports that the puzzle has no solution. For example, given the puzzle in the following figure, your algorithm would return the number 5.

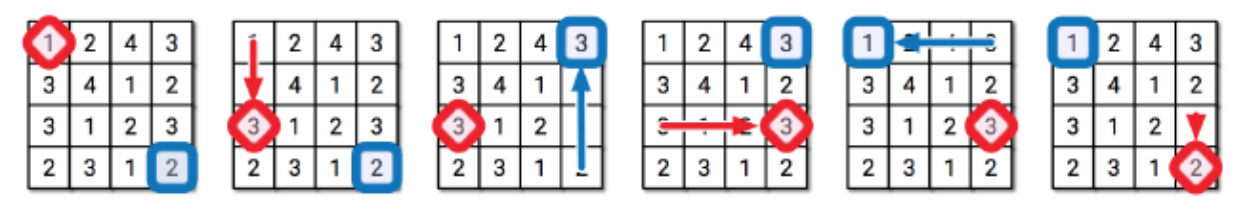

Figure 1: Example solution

Description

In order to solve this problem efficiently, we implemented an iterative breadth-first search (BFS). The BFS is initialized as a FIFO queue of system states. The states take the format of a tuple of the coordinates of the red and blue pieces on the board, e.g. the initial state is $((0,0),(n-1,n-1))$ where n is the board size. The second component of our algorithm is a dictionary of the moves needed to reach each state, indexed by the state tuple. This allows quick $O(1)$ access for both checking if a state has been visited and how many moves were needed to get there.

As we iterate through the queue, all possible moves from the current state are checked. For every state possibility, there will be 8 possible moves from the current state: moving red or blue up, down, left, or right. For each of these moves, we check if the new state is valid: that is, the new position is in bounds, not occupied by the other piece, and is not in the dictionary. If that state is indeed valid, we add it to the end queue and record the moves needed to achieve that score in the dictionary.

If the end state is reached – where the two pieces have swapped starting positions – we

Any states we have already discovered will be ejected from the BFS queue, and marked so repetition is avoided. The goal of the game is to reach the state (n-1, 0) as fast as possible, so once this state is reached within any of the branches of the BFS, the move count should be immediately returned. This move count will be the minimum number of moves that is required to go from the initial state described in the problem, to the final game-winning state.

Runtime Analysis

We know that the runtime of a BFS is O(V + E), where V is the number of vertices, and E is the number of edges. In this case, the BFS is essentially creating a partially-completed graph of the state space, as it doesn't require every single possibility, but rather the states required (and some change) to reach the final end state via the actions taken. The worst case number of verticies is $n^4 - n^2$, or n^4, where n is one of the sides of the grid. This is because the total number of possible squares for the red token to be in is n^2, and for the blue token is also n^2, and they cannot be in the same square at the same time, resulting in a $-n^2$. Similarly, the number of edges are $c * n^4$, where c is an arbitrary constant. The edges of the BFS are organized and aren't cyclic, so there are a maximum of 4 connections per vertex, and if there are n^4 vertices, then there are $4 * n^4$ edges. In terms of time with respect to n: $O(n^4 + 4n^4) = O(n^4)$. Therefore, the time for this algorithm is $O(n^4)$.

Proof of Correctness

First, we need to describe the mechanics of our BFS, using these two statements:

1) A path of length k in the graph gives a winning strategy with k moves.
2) A winning strategy with k moves gives a (0, n-1) -> (n-1, 0) path of length k.

These statements, along with the fact that BFS is proven to give the shortest path, prove that the minimum length of the graph gives us the minimum number of moves to win the game.