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.

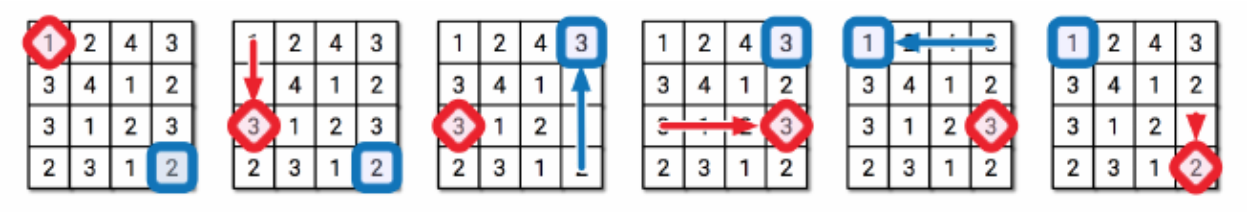

Figure 1: Example solution

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 have reached the shortest path to the end state and record the moves needed to get here from the dictionary. If the queue is exhausted, all valid states have been visited without reaching the end node and we return -1.

Runtime Analysis

The runtime of a breadth-first search (BVS) is $O(V + E)$, where V is the number of vertices and E is the number of edges. In this case, the BFS creates a partially-completed graph of the state space, as not every possibility is required – only the ones needed to reach the final end state. The worst-case number of verticies is $n^4 - n^2$, or n^4, where n is one of the sides of the grid. The total number of possible locations for one piece on the board is n^2. For two pieces, the number of states is squared for n^4. As the pieces cannot occupy the same square at the same time, n^2 possible states are removed. Similarly, the number of edges are $c * n^4$, where c is an arbitrary constant. The edges of the BFS are organized and are nott cyclic, so there are a maximum of 4 connections per vertex. For n^4 vertices, then there are $4 * n^4$ edges. All together, the runtime of this algorithm is $O(n^4 + 4n^4) = O(5n^4) = \mathbf{O(n^4)}$.

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 $((0,0),(n-1,n-1)) \rightarrow ((n-1,n-1),(0,0))$ path of length k.

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