Group Assignment 3

Jacob Cheney, Brandon Lam, Philip Warton

November 17, 2020

1 Algorithm Description

We use a breadth-first-search graph algorithm to solve the Vidrach Itky Leda puzzle. Define a state to be an ordered pair of points, the first being the position of the red token, the second being the position of the blue token. Our set of vertices $V = \{\text{all possible game states}\}$, and our set our set of edges to be all legal moves that move the configuration of the board from one state to another. This graph will be unweighted since every edge has the same "length" of being one move. This graph is also undirected, since for any move, one can simply perform the reverse.

Let n be the number of both rows and columns on our game board. Let $v_0 = ((0,0),(n-1,n-1))$ be our initial game state as defined by the game rules, and let $v_f = ((n-1,n-1),(0,0))$ be our final game state. We must answer two questions:

(i) Is v_f reachable from v_0?

(ii) What is length of the shortest path $v_0 \to v_f$?

To answer this, we use a queue to perform a breadth-first-search algorithm, keeping track of the shortest distance $d(v_0, v)$, where $d : V \times V \to \mathbb{Z}^+$ is the shortest distance between two vertices. For each vertex v in the queue, we add each unvisited vertex v' to the queue, and we mark $d(v_0, v') = d(v_0, v) + 1$. We either return $d(v_0, v_f)$, or if our queue becomes empty before this is possible, we return -1 as an indication that v_f is not reachable from v_0 (i.e. there is no set of legal moves that gets us to the winning position). The general structure of the algorithm is the following:

```
function SolvePuzzle($G$, $v_0$, $v_f$)
    Queue = $\{v_0\}$
    Distance[$v_0$] = 0
    Visited[$v_0$] = False
    while Queue $\neq \emptyset$
        $v$ = Queue.pop
        while $v \in$ Visited
            $v$ = Queue.pop
        for $v'$ reachable, unvisited from $v$
            Parent$[v'] = v$
            Distance$[v']$ = Distance$[v] + 1$
            Queue.push$(v')$
        if $v = v_f$
            return Distance$[v]$
    return $-1$
}
```

2 Running Time Analysis

We use a while loop upon the queue until it is empty, and since we could potentially pass through every possible configuration before arriving at the final winning position, this occurs in at least $O(V)$ where V is the number of possible configurations.

Then within this while loop we perform several tasks, but claim that each occurs in constant time. For each of the 8 possible moves that can be done from some given position, we can check if we have visited the target state, and check if it is a valid state both in $O(1)$ time. Then since there are 8 we say $O(1) \cdot 8 = O(1)$ still. We append each of the remaining states to our queue, which by using a double ended queue we can do so in $8 \cdot O(1) = O(1)$ time as well. To compute the distance of each reachable vertex, we simply access $d(v_0, v)$ from a dicitonary in $O(1)$ time and add 1 to it. We do this 8 times also. Since 8 does not rely upon the input size, we say these actions are all within constant time.

Thus the while loop is performed in $O(V)$ time, and we need only now to perform a return, which occurs in constant time. We say $O(V) + O(1) = O(V)$ clearly.

To relate this to our input size n, we must compute how many possible game configurations exist. We have a total of $n \times n = n^2$ tiles on the game board. So for the red token, we can choose from n^2 positions, and for the blue token we can choose from the remaining $n^2 - 1$ positions. This gives us a total of $n^2(n^2 - 1) = n^4 - n^2$ combinations. This number of possible states is equal to the number of vertices on the graph, so we can translate our runtime as follows:

$$O(V) = O(n^4 - n^2) = O(n^4)$$

3 Proof of Correctness

We assume without proof that a BFS algorithm provides the shortest path $s \to v$ in a graph. We need only to prove then that there exists a direct correspondence between this graph and our puzzle, and that the shortest path on the graph provides the shortest set of moves in the puzzle.

Proof. Define

$$S = \{(R, B) \mid R \neq B \text{ and } R_x, R_y, B_x, B_y \in 1, 2, \cdots, n-1\}$$

to be the set of all game states. Then define

$$M = \{\{s_1, s_2\} \mid s_1, s_2 \in S \text{ and there exists one legal move between them}\}$$

to be the collection of all legal moves in the game. Then it is simple to define some graph $G(S, M)$ that takes the set S as the set of vertices and the collection M as the set of undirected, unweighted, edges.

Assume that there exists some shortest set of moves that solve the puzzle,

$$s_0 \to_{m_1} s_1 \to_{m_2} s_2 \to_{m_3} \cdots \to_{m_k} s_f$$

Assuming that s_f is reachable from s_0, this path will have some finite length $k \in \mathbb{N}$. Suppose that the BFS algorithm produces some path with a length k'. If $k' < k$, then it follows there must be some set of legal moves corrosponding to each edge such that this solution is shorter than our 'shortest set of moves' (contradiction). If $k' > k$ then the set of edges corrosponding to $(m_1, m_2, \cdots, m_k)$, will be a path from $s_0 \to s$ that is shorter than the one provided by the breadth-first-search algorithm. Therefore the BFS algorithm does not provide the shortest path (contradiction). So it must be the case that $k' = k$ and this algorithm provides the shortest set of possible moves to a reachable winning state s_f.

If there is no solution, the queue will become empty, and the BFS algorithm will produce no path. Having kept, track of all visited vertices, we can clearly notice that s_f is not reachable, and return the correct result -1. □