# CS 454 THEORY OF COMPUTATION FALL 2018 FINAL PROJECT REPORT

**Team Members:**

Ariel Katz

Darin Brown

Jack Weatherford

# Problem Statement

6. Design a NFA to accept the set of solvable strings in the case of 3 by n slitherlink for all n. (Search for slintherlink and play it to understand the rules.) The input in this case will be over the alphabet S = {[a, b] | a, b are in {0, ... , 4}}. Then write a program to simulate the NFA and output a solution (not just yes/no answer). The following figure shows an input string w = [3, -1][-1, 2][-1, 3][1, 2][3, -1] that encodes the puzzle shown in the figure above. In the encoding, -1 stands for unconstrained. i.e., the number of segments that can be drawn around them can be any number between 0 and 4. One possible output will be the image as shown below:

The idea is to use a lazy technique to find solutions to this simple game.

**NOTE:** ​This algorithm does ​**not** ​work as intended

The idea is that given a graph, we can infer the structure of the DFA associated with it, by trimming cycles. We identify our environment through experimentation and can ensure the accuracy with a calculated margin of error. We generate a strongly connected connected graph, where every vertex has the same in and out degree with cycles removed. For the purpose of a rubik’s cube, this means every vertex has 3 edges going into it, and 3 edges coming out of it. To ensure that our algorithm is generating correct outputs.

# Background

Our program is designed to create states for board configurations at the level of a single column of the board which we call a cell. This is comprised of two mini-cells, each with a specified constraint (-1, 0, 1, 2, 3). As the program runs, it creates a new state for each possible cell constraint as it is encountered. A state is defined by the point the snake enters the cell, the point the snake leaves the cell, and the mini-cell numerical constraints.

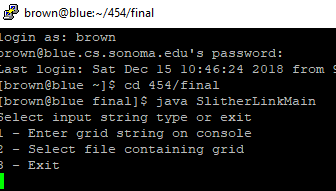
Example Board with a highlighted cell state

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | -1 | 0 | 2 | -1 | -1 | 2 | 3 |
| 1 | -1 | 2 | 0 | 1 | 2 | 0 | -1 |

Three possible entry points A Cell 3 possible exit points

Our program checks the possible configurations that meet the constraints of a state and adds it as a possible path. It then moves on to the next cell and attempt to use a configuration for the next state, continuing until it encounters a state to state transition with no solution. It then backtracks and tests the next valid cell configuration that meets the state constraints. It prints the paths to console as it traverses the board. If it cannot find a sequence of states to reach then “end” cell, it advises “no solution”.

# Methodology

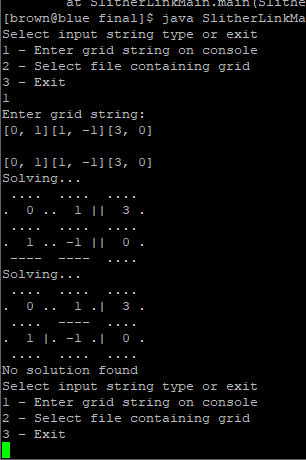
***UI:***

***Input: Output:***

*A text file or console input A path printed to console or*

*in the following format a “No Solution” message*

*[3, -1][2, 0][1, 1][-1, 2] see below*

**

**Time Complexity**

Since the algorithm uses dynamic programming based on possible states of cells, it is not always dependent upon the input size (n). Instead, in the worst case, it is bounded by the number of possible configurations (c) (constraints + output options) \* a maximum of c checks for a solution \* a maximum of c configurations hashed. It is extremely unlikely that the algorithm would have to do the maximum number of checks for each state, so this upper bound of c^3 will be reached in an infinitesimal number of situations. In fact, the number of valid cell configurations for a given set of constraints is in reality between 0 and 17, with most around 3 and an average of 4. This dramatically decreases the time complexity of the algorithm. The upper bound then becomes O(n\*c\*5^2). Where k is 25c (c is on the order of 10^2), the worst case complexity is in O(nk) or O(n).

**Conclusions**

If we could have gotten this algorithm to work, it would be extremely efficient for large inputs. For inputs < 2500, it is in O(n^2) for inputs close to 2500 to O(n^4) for very small inputs. The challenges we found were mapping out all possible states and there is a minor problem we haven’t been able to track down that prevents the sequential solving to work properly.