Mitchell Brown
Analysis of Algorithms
March 4, 2021

# Portfolio Project

**Part 1: Introduction to Nurikabe**
I chose to implement the Nurikabe puzzle game, number 13 from “A Survey of NP-Complete Puzzles.” This game is played on an n x m grid, of which I chose 5x5, and within this grid there are n-1 (n being 5 for my game) digits. These digits represent the number of squares that are to be left unshaded adjacent to themselves. The goal of the game is to shade cells such that they meet a set criteria:

> “1. Each partition of cells is contiguous.
> 2. No numbered cell can be shaded.
> 3. There cannot be any 2 × 2 blocks.
> 4. Each contiguous block of white cells only contains one numbered cell and that each of these blocks contains a number of cells equal to the value given in the numbered white square within that block.”
> From “A Survey of NP-Complete Puzzles,” page 21-22.

**Part 2: Proof of Correctness of Verification Algorithm**
For my puzzle, I chose to use a BFS to verify the user’s solution. Since there are a few criteria to meet, there are two instances of BFS: first to check the shaded region, next to check the unshaded regions. This is done to ensure all regions are contiguous. The shaded BFS, in addition to checking contiguity, also checks for any 2x2 blocks. Therefore, all criterium are met from the goals of the puzzle.

Next, it is important to prove my implementation is correct. First, let’s evaluate the validity of checking the unshaded regions. This implementation has a list of each value of regions to be left unshaded. It goes through these, and performs a BFS starting at the value. Figure 1 shows an example of this BFS being performed on a correct solution, starting from the value “5.”

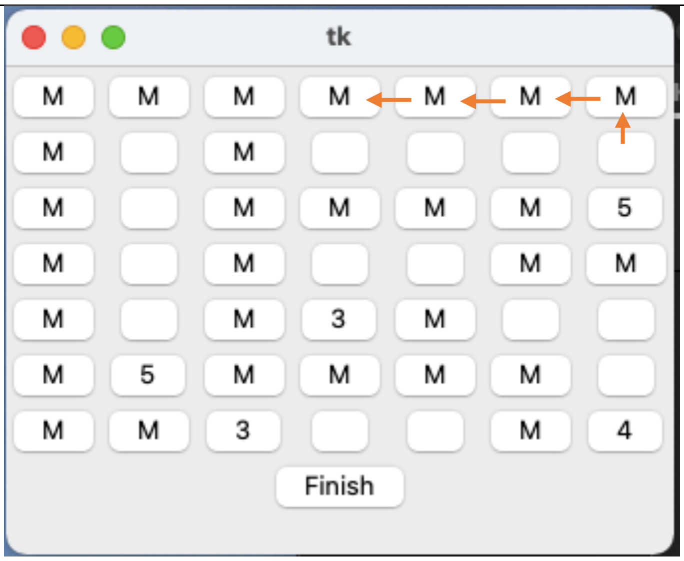
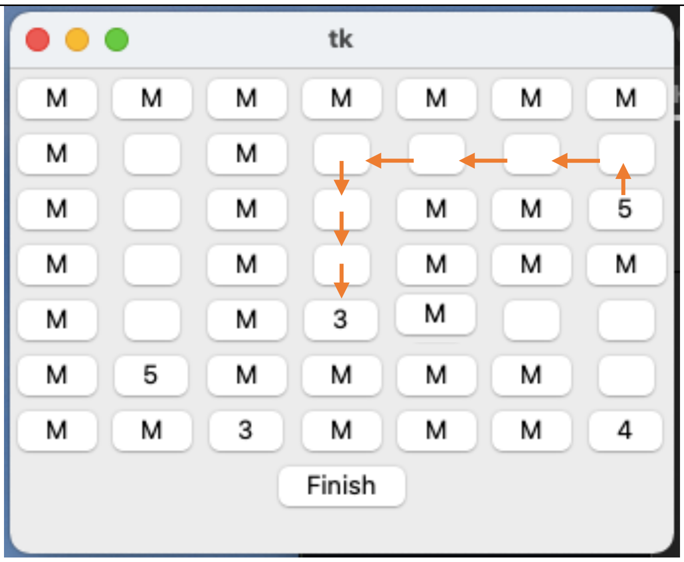


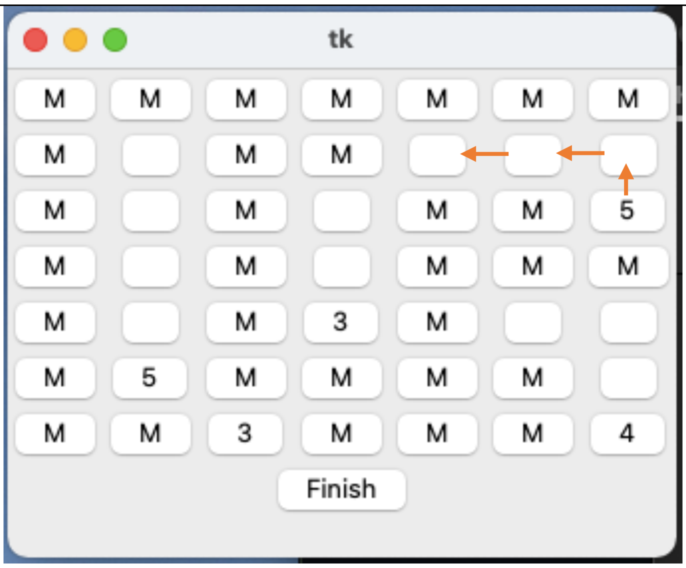
**Figure 1. Example of BFS on Unshaded Regions**

In Figure 2, we can see the same BFS being performed on an invalid solution, starting from the same value of "5." In this example, we see the regions of "3" and "5" are connected, violating rule 4 of the game, in which each unshaded region must not connect to another. When the BFS is performed on the value "5," it will see the region size is greater than 5, and state the invalidity. In figure 3, we see another example of the unshaded region being invalid, being the same region for "5" only has 4 cells unshaded. The BFS will catch this error in the same fashion, by comparing the size of the region (4) to the expected (5).



| | | | | | | |
|---|---|---|---|---|---|---|
| M | M | M | M | M | M | M |
| M | | M | | | | |
| M | | M | | M | M | 5 |
| M | | M | | M | M | M |
| M | | M | 3 | M | | |
| M | 5 | M | M | M | M | |
| M | M | 3 | M | M | M | 4 |

**Figure 2. Unshaded Regions Connected**



| | | | | | | |
|---|---|---|---|---|---|---|
| M | M | M | M | M | M | M |
| M | | M | M | | | |
| M | | M | | M | M | 5 |
| M | | M | | M | M | M |
| M | | M | 3 | M | | |
| M | 5 | M | M | M | M | |
| M | M | 3 | M | M | M | 4 |

**Figure 3. Unshaded Region too Small**

Once the BFS has been performed on all values, the unshaded regions need to be checked for two things; contiguity and 2x2 blocks. First, let's look at how I handled checking contiguity. The easiest way to evaluate this is to run a BFS on the shaded blocks, then count up how many are shaded, and compare that to the expected. If they do not match, it means one of two things: first, the incorrect number of cells has been shaded, and second, the shaded cells are not contiguous. The first of these checks will already be handled by the BFS on unshaded cells, so once we get here and find a mismatch, we will know it is due to a discontiguity. Figure 4 shows two examples of a BFS, the first being a correct solution (Figure 4.1), the second being an incorrect solution (Figure 4.2).

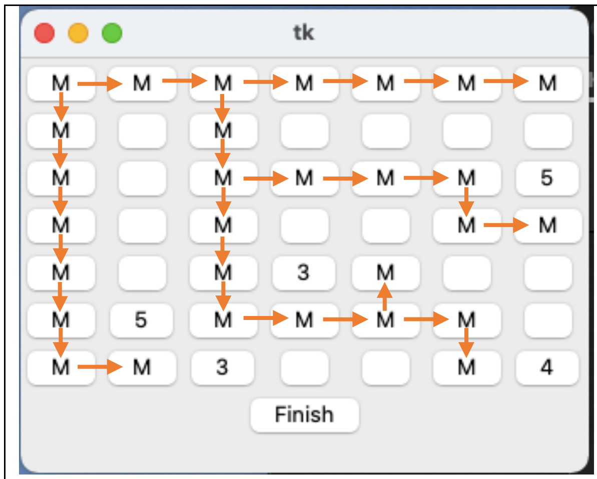


**Figure 4.1. A BFS on a Correct Solution**

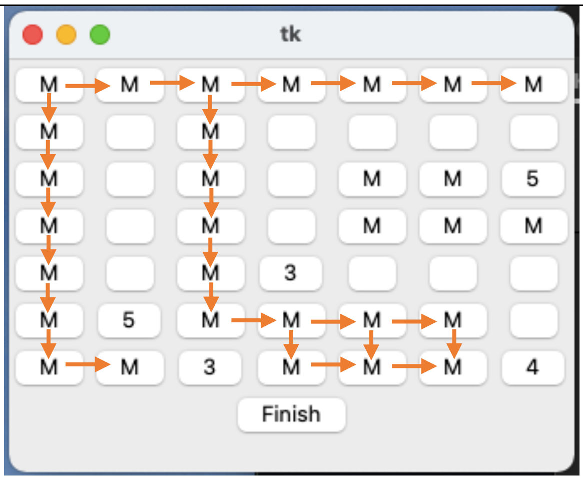
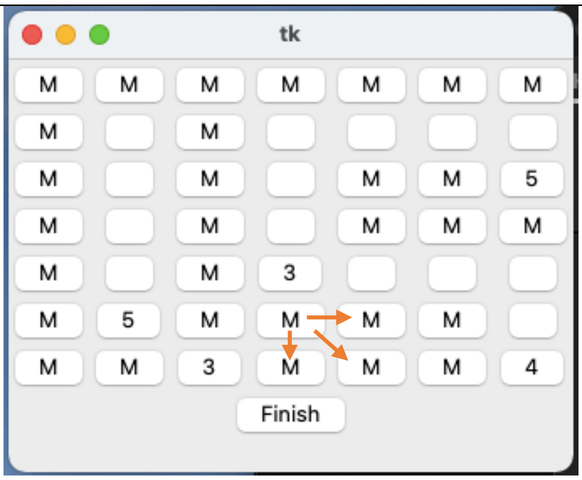


**Figure 4.2. A BFS on an Incorrect Solution**

**Figure 4. Running BFS on the Shaded Regions**

The other item checked for in this BFS is for a 2x2 square. This is checked by moving to the three squares adjacent to the node being evaluated. In other words, if [0, 0] is being evaluated, it will look at [1, 0], [0, 1] and [1, 1]. If all three are marked, it means there is a 2x2 square. You may notice this was not checked for in unshaded regions; this is because if a puzzle setup is valid, there will never exist a 2x2 unshaded square without there also being a 2x2 shaded square or some other invalid setup. Figure 5 shows an example of a node being evaluated where the search will find a 2x2.



| | | | | | | |
|---|---|---|---|---|---|---|
| M | M | M | M | M | M | M |
| M | | M | | | | |
| M | | M | | M | M | 5 |
| M | | M | | M | M | M |
| M | | M | 3 | | | |
| M | 5 | M | M | M | M | |
| M | M | 3 | M | M | M | 4 |

**Figure 5. Detecting a 2x2 Block**

**Part 3: Time Complexity**

The time verification algorithm utilizes multiple instances of BFS (6 in the example puzzle instance). BFS grows at a rate of O(VE), where V is the number of vertices and E is the number of edges. In this implementation on an n x n board, there are $n^2$ vertices, and there are 2(n)(n-1) edges, which is $2(n^2 - n)$. Therefore, the overall time complexity is $O(n^2 * 2(n^2 - n))$, which simplifies down to **$O(n^4)$**, where n is the length of one side of the board.