Casey Cheek
CS325 - Portfolio Assignment

Pick one of the 24 puzzles you will work on in this assignment. Mention which puzzle did you pick?

I chose to do the KPlumber puzzle.

Write the rules of the game for the puzzle you picked.

The rules of KPlumber, as described by Kendall, Parkes, and Spoerer (2008), are as follows:

> KPlumber is a computer puzzle game played on an m × n grid of tiles. Each tile in the interior of the grid has four adjoining tiles, the corner tiles have two adjoining tiles, and the edge tiles have three adjoining tiles. At the centre of each tile is an intersection of up to four pipes, each of which runs directly to one of the four sides of the tile. Some tiles have no pipes. Water runs through the pipes and leaks from any that are left open. An open pipe at the edge of a tile that touches an open pipe at the edge of the adjacent tile closes both of the pipes. An action rotates one tile, and the pipes on it, by 90◦. The goal is to arrive at a situation in which all pipes are closed so that there is no leaking water. This is known as a safe state.

Implement a program that allows the user to solve the puzzle. The puzzle should be in its standard size, if any. For example, if you choose Sudoku, the size should be 9X9. If the puzzle has no standard size, please choose any of the known ones for that puzzle.

My program, kplumber.py, allows a user to play KPlumber with a CLI (see README.txt for details).

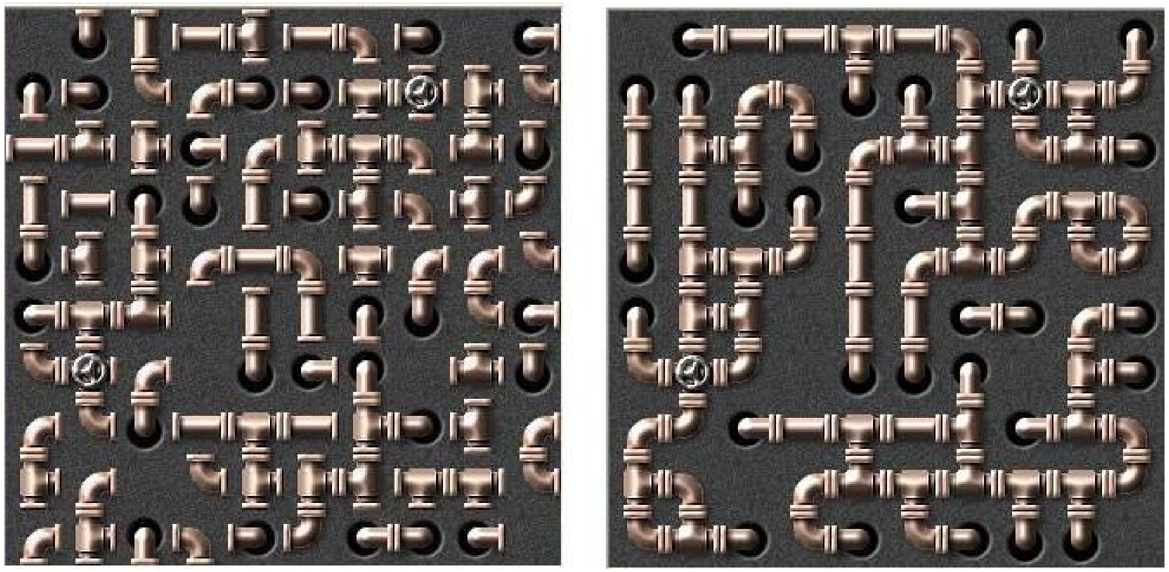

Figure 1. Unsolved (left) and solved version (right) of the same instance.

Kendall, Parkes, and Spoerer (2008) describe a version of KPlumber called “Linkz” created by Martijn Van der Lee. They go on to show the solved and unsolved states of a 10X10 “Linkz” instance (Figure 1). I used this example to code two input files: one representing the unsolved version (called kp1.txt) and another representing the solved version of the same instance (called kp1_solved.txt).

Include in your program an algorithm that verifies the user's solution to the puzzle. The algorithm takes the solved version of the puzzle board as input and verifies the solution. Your algorithm should return "Solved!" (or similar text of your choice) if the input board provided is solved correctly, else return "Not Solved, Try Again!" (or similar). Include a README file with your code

KPlumber has a function called verify() which reports if the current state of the game board is solved or not. A user can try this function out by running the kplumber.py program, then typing the command, “verify” (see README.txt for details). If it is solved, verify() will return “Solved!”. Otherwise, verify() will return “Not solved. Try again.”

Prove the correctness of the algorithm that you wrote to verify the solution.

Kendall, Parkes, and Spoerer (2008) describe the solved state as “a situation in which all pipes are closed so that there is no leaking water.” The verify() algorithm works by examining each side of each tile on the board. If a side of a tile is “open” then it performs two checks:

- Is that side open to the edge of the board, or...
- Is the tile adjacent to that open side closed where the two tiles touch?

If either check produces a “yes” for **any** open tile side on the board, then the puzzle is not solved. This tile represents a leaky pipe. Otherwise, that means that **all** tile sides fall into one of two categories: the side is closed or the side is open and adjoins the open side of an adjacent tile. In other words, there are no leaks and the puzzle is solved.

Discuss the time complexity of the verification algorithm.

We can find the number of tiles on the board *n* by multiplying the number of rows *r* by the number of columns *c*. Verify() checks each of the 4 sides of every tile on the board. Checking one side of one tile is done in constant time. If any check fails, verify() immediately stops searching and reports that the puzzle is not solved. So in the worst case, where verify() must check every tile side to find an answer, the number of checks that must be performed is $r * c * 4$ or $4n$. Thus, its running time is $O(n)$. This confirms that KPlumber is verifiable in deterministic polynomial time.

As a side note, verify() could likely be altered to run even faster. There are a lot of repeated checks in the current design that could be made more efficient with dynamic programming or memoization.

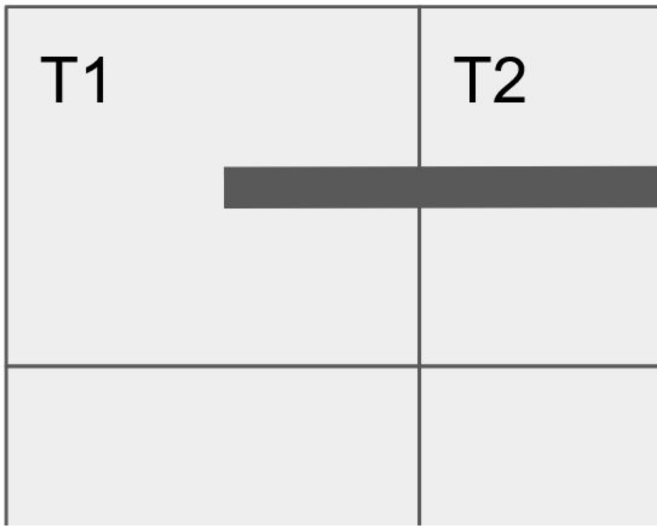

Figure 2.

In Figure 2 for example, we must perform 4 checks on the first tile t_1, during which we confirm that the open right side of t_1 lines up with the open left side of t_2. When we move on to examine t_2, we don't need to check each of its 4 sides. We already know that the edge between t_1 and t_2 is in a "safe state".

References

Kendall, G., Parkes, A., & Spoerer, K. (2008). A survey of NP-Complete puzzles. ICGA Journal, 31(1), 13–34. https://doi.org/10.3233/icg-2008-31103

Van der Lee, Martijn (2009). *Linkz*. VanDerLee.com. http://www.vanderlee.com/linkz/index.html