**Pipe Traverse Tool (traverse\_v4.exe)**

Author: Bill Thaman ([bill.thaman@hdrinc.com](mailto:bill.thaman@hdrinc.com))

Date: 2/21/2022

**Purpose of Tool**

For each pipe in a pipe network (e.g., wastewater collection system network), recursively traces the upstream pipes and calculates the total length and count. The tool will fail if there are any loops in the system.

**Structure and Execution**

Three files are required to run the tool:

1. traverse\_v4.exe – Tool’s executable.
2. traverse\_v4.toml – Configuration file containing the path to the input file.
3. network.csv – Comma-separated values (csv) file containing the pipe (edge) identifiers, the upstream node (e.g., manhole), the downstream node, and the length of the edge. The base name can be anything if it’s a .csv extension with the correct columns in the correct order. To run the tool using a particular input file, enter the path to the input csv in traverse\_v4.toml.

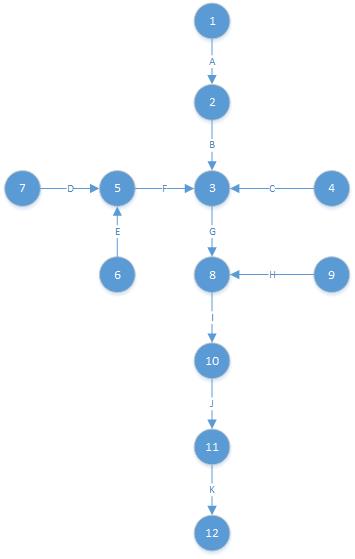
The preferred method to execute the tool is to open a command window, navigate to the folder containing the three files listed above, and type “traverse\_v4.exe”. This method ensures that the command window remains open after execution is complete. Other methods include executing from a Windows batch file, double-clicking traverse\_v4.exe in Windows Explorer, or executing from a script or application.

Upon successful execution, three csv files are output:

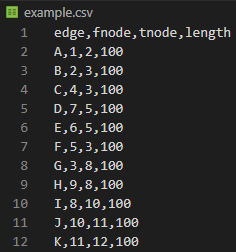
1. upstream\_pipe\_count.csv – Count of upstream pipes for each pipe
2. total\_length.csv – Total length of upstream pipe for each pipe
3. edges.csv – List of upstream pipes for each pipe

**Example**

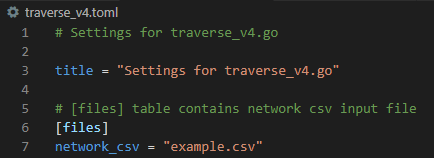
A sample network is shown below. All pipe lengths are 100 ft.



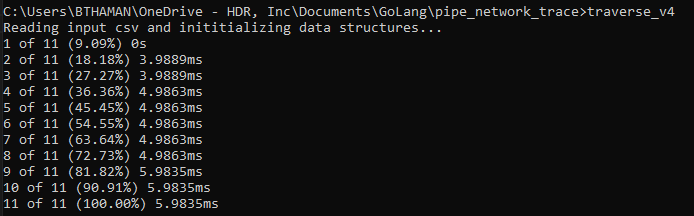
Input file (example.csv):



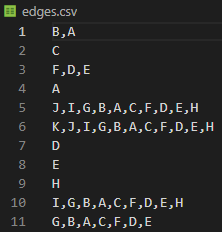
traverse\_v4.toml is shown below. If the .exe and .toml file are in the same directory the full path to the input file is not required.



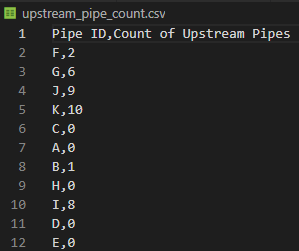
Execution command window shown below. Execution time is shown in the last column.



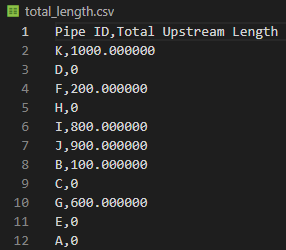
edges.csv shown below. Column 1 is the edge in question, and columns 2-n are the upstream edges.



upstream\_pipe\_count.csv



total\_length.csv:



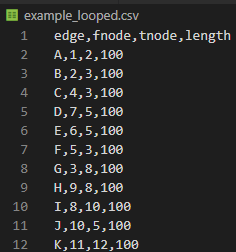
**Handling Loops**

A loop in a collection system is typically caused by incorrect coding of the upstream (fnode) and downstream (tnode) nodes. As a demo, a loop is introduced to the example network as shown by the dashed line below. It’s shown as a dashed line for illustrative purposes; the actual network obviously will not show the loop graphically.

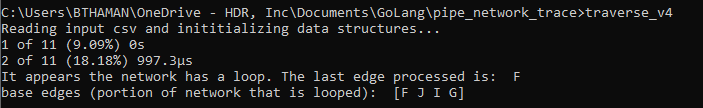
Chart

Description automatically generated

example\_looped.csv shown below. The loop is introduced by changing the downstream node of edge J to 5.



The execution is shown below. As expected, execution failed.



For this simple example, it is not difficult to spot the problem by looking at the input file and the network graphic. In a real network with tens of thousands of pipes, it can be challenging to find the loop. The key to finding the loop is to look at the error message that lists the edges that form the loop: in this case J-F-G-I (order shown in error message may not be in order of flow). By looking at the edges in the loop you can determine where the bust is.

**Performance**

The tool’s programming language is Go, which provides excellent performance on par with C, Fortran, and Java. Another key to performance is only tracing a given path once, but this should be done irrespective of programming language.

As an example of what to expect, traverse\_v4 traced a 31,917-pipe network in 85.4 seconds (374 pipes per second). Note that Go uses hash tables to store arrays and does not guarantee that the elements in an array are stored in any particular order, so the order may be different each time the program is run. Because of this, the execution time will be different each time, but should never be less than about 200 pipes per second in a large pipe network.

