Columbia River Binary Tree Project Report

Carson Frost, Joe Mock, Charlie Serafin, Jessica Sun

1. Introduction

The Columbia River, one of the most significant natural features of the Pacific Northwest, is supported by a vast network of tributaries and controlled by numerous dams. This project models the Columbia River and its watershed using a binary tree structure in C++. It enables traversal of river features, addition of new tributaries and dams, and outputs key information.

Running the Program:
g++ main.cpp RiverTree.cpp -o program
./program.exe

Running Unit Tests
g++ unit_test.cpp RiverTree.cpp -o test
./test.exe

2. Initial Design and Changes

Initial Design Choices:

- Binary Tree Structure: Each node has up to two children.

- Right child nodes are tributaries of their parents.

- Left child nodes are **continuations of the parent** (only if there is a right node)

    - This was intended to keep 'rivers' continuing along a path (leftwards) with tributaries branching to the right

- `Tributary` and `Dam` structures encapsulate relevant attributes.

- Attributes: length, basin size, discharge

    - Note: fake values are used (roughly approximated), mostly used to show the opportunity for information storage

- `RiverNode` objects store either a tributary or a dam, never both, and point to children.

- Root Node: Represents the Columbia River's mouth and serves as the tree's entry point.

Design Changes:

- Separate Constructors for Dams and Tributaries in `RiverNode`. This change was made because information for a tributary vs. a dam is different.

- Manual vs. Dynamic Input: Originally intended user input for node creation; final implementation shifted toward hardcoded data. This is obvious now, but wasn't to us initially, that making the user input data every single didn't make sense long term.

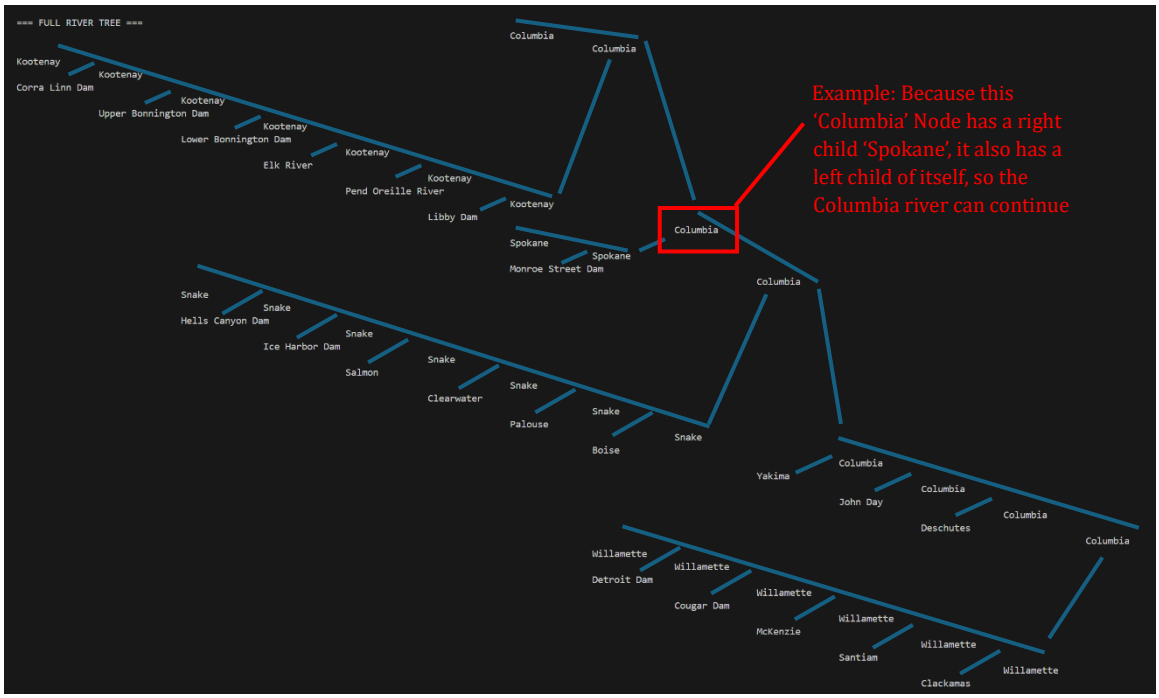
- Structure of the tree. Initially, we had tributaries as right children, and dams as left children. We opted for this design because frankly, it makes the tree look really cool.

3. How to Use

A simple menu should make using the program fairly intuitive, but here are the basic features:

1. Print full tree.
    - Prints the full tree, with root furthest to the left, and leaf nodes furthest to the right. This allowed us to use a basic in-order traversal structure for printing, where leftmost nodes were printed first and rightmost printed last.

Example of printed tree, with lines to help make format clear:



2. Explore tree.
    - User will be presented one node at a time, with all relevant information, they can then move from node to node along connections.
    - Note: In the code, this feature initializes the parent_node, which is an odd place to do this. In the future this could be cleaned up, but for now, this was the only feature that needed a node to have a pointer to its parent.