# 1 Goals

- Practice with basic balanced binary search trees;
- More practice with linked structures, pointers, and dynamic memory management.

Note that you may use whatever environment you like for this class, but your programs must be able to compile and run using `g++` (or `clang++`), `cmake`, and `make`. For this assignment, you will also need `valgrind` to check for memory errors and leaks as well as `gnuplot` for generating performance graphs. You may also find it helpful to have `gdb` for helping to debug segmentation faults. The department provides a remote development server (`ada.gonzaga.edu`) running Ubuntu that can be accessed using an ssh remote connection from within VS Code on your own computer. The remote server contains all of the tools you need for assignments in this class. Depending on your computer's configuration, you may be able to install the tools you need locally to complete the homework assignments. It is important that you start assignments early in this class so that if you have questions you can ask them and get them answered with enough time before the homework deadline (to avoid late penalties).

# 2 Instructions

1. Use `git clone` to clone the classroom repository created for you and to obtain the starter code (either onto the remote development server if using `ada` or locally if you are using your own machine). Be sure to frequently add, commit, and push your updated files back to your GitHub repository (via `git add`, `git commit`, and `git push`).
2. Copy your `sequence.h`, `map.h`, `arrayseq.h`, `binsearchmap.h`, `hashmap.h`, and `bstmap.h` files from your HW-8 repository. These **must** be present in your repository for me to grade your submission. (That is, I have to be able to compile your code to run and test it.)
3. Implement the functions declared in `avlmap.h`. Note that many of these will be identical to those in `bstmap.h`. See the comments and lecture notes for details.
4. Make sure your implementations pass the basic tests provided in `hw9_test.cpp`. You do not need to write any additional unit tests for this assignment (however, you are encouraged to write additional tests as you see fit). Note the tests provided are in no way comprehensive.
5. You must run `valgrind` on the `hw9_test` program to ensure it does not detect any memory issues in your implementations. You should run `valgrind` once you get the unit tests to pass. If your program has memory issues as reported by valgrind, you will only receive partial credit on the assignment.
6. Once your code is completed, all of the unit tests pass (including your new tests), and you do not have memory issues as reported by `valgrind`, run the provided performance tests in `hw9_perf`. The performance tests will generate a data file with testing results, which you will then graph using `gnuplot`. You will need to add the corresponding graphs to your assignment write up (next step). *Note that running* `hw9_perf` *to completion should take up to around 1 minute to complete, which is longer than in previous assignments.* If the performance tests take longer than this for you, there is likely something wrong with your implementation(s).

7. Create your assignment write up and add it as a PDF file to your assignment (GitHub) repository. You **must** save your writeup as a PDF file and name it `hw9-writeup.pdf`. (Note no capital letters, no spaces, etc.) Be sure to push all of your source code and your write up by the due date so it can be graded. (Note that you can check that everything is in your repo from the GitHub website and/or using the `git status` command.) See below for expectations concerning your assignment writeup. Once you have submitted your files to your GitHub repository and are ready to submit your assignment for grading, you must fill out the grading submission form. A link to the form will be provided in piazza (in the same post as the GitHub classroom link).

## 3 Additional Requirements

Details for HW-9 were discussed in class and are provided in the lecture notes. For this assignment you may not add any additional helper functions and must use the techniques discussed in class to implement each function. Note that this primarily means using iteration versus recursion. Those functions that should be implemented recursively have recursive helper functions declared.

**Left and Right Rotations.** In this assignment, the height of each node in the tree is stored in the node itself. Left and right rotations are performed within the rebalance function. You can adjust the heights of associated nodes after a rotation either within the rebalance function or in the rotation functions themselves (preferred).

**AVL Statistics.** The height function (like in HW-8) is used in the performance tests to determine the height of the associated trees created. The height is then compared to $\lceil \log_2 n \rceil$ (which can be viewed as an "ideal" height). For this assignment, your AVL Map height function should return values very close to $\lceil \log_2 n \rceil$ (typically one value higher than this).

**Homework writeup.** Your homework write up must contain each of the graphs produced by the performance tests together with an explanation as to why you think the performance tests came out the way they did based on what you know about the implementations. Finally, briefly describe any challenges or issues you faced in completing the assignment.