I&C SCI 46 Lecture Fall 2021 Project 4: Counting the Trees

Due **November 18, 2021, 9:59 AM**

Checkpoint is due November 12 at 9:59 AM.

There is a “checkpoint” portion: you **must** submit your partial code by November 12 at 8:59 AM. The first four test cases provided, marked as checkpoint, will be the only four graded for this checkpoint. This is worth one-sixth of your “correctness” grade for this assignment.

Introduction

Do you know about Zipf’s Law? It relates to a phenomenon for how very frequent items can sometimes appear far more frequently than less frequent items. For example, the word “the” is something like 7% of words when an appropriate corpus of English-language text documents are examined.

If you were interested in examining Zipf’s Law, you might write a computer program to read a bunch of text documents and count how often each word appears. Depending on various factors, one data structure you might consider using is a balanced binary search tree.

Getting Started

Before you begin work on this project, there are a couple of chores you'll need to complete on your ICS 46 VM to get it set up to proceed.

Refreshing your ICS 46 VM environment

Even if you previously downloaded your ICS 46 VM, you will probably need to refresh its environment before proceeding with this project. Log into your VM and issue the command ics46 version to see what version of the ICS 46 environment you currently have stored on your VM. Note, in particular, the timestamp; if you see a version with a timestamp older than the one listed below, you'll want to refresh your environment by running the command ics46 refresh to download the latest one before you proceed with this project.

If you're unable to get outgoing network access to work on the ICS 46 VM — something that afflicts a handful of students each quarter — then the ics46 refresh command won't work, but an alternative approach is to download the latest environment from the link below, then to upload the file on to your ICS 46 VM using SCP. (See the Project #0 write-up for more details on using SCP.) Once the file is on your VM, you can run the command **ics46 refresh_local NAME_OF_ENVIRONMENT_FILE**, replacing **NAME_OF_ENVIRONMENT_FILE** with the name of the file you uploaded; note that you'd need to be in the same directory where the file is when you run the command.

Creating your project directory on your ICS 46 VM

A project template has been created specifically for this project, containing a similar structure to the basic template you saw in Project #0.

Decide on a name for your project directory, then issue the command **ics46 start YOUR_CHOSEN_PROJECT_NAME project4** to create your new project directory using the project3 template. (For example, if you wanted to call your project directory proj4, you would issue the command `ics46 start proj4 project4` to create it.) Now you're ready to proceed!

Choosing a project partner

You *have the **option*** to work with a second person for this assignment. If you do so, I expect you to work via pair programming. That is, you may **not** split the assignment, such as by having one person implement the AVL Tree while the other person implements the function that does the counting, and the two are stitched together later. I reserve the right to ask one or both project partners about the implementation and adjust the score accordingly.

Similarly, any academic dishonesty arising from a group will be treated as an offense by both partners.

To declare a partnership, **both partners** need to fill out the following form by Wednesday, November 11 at 9:59 **PM**. *There will be no exceptions granted. Be sure you fill it out correctly and that you know what your UCINetID is and how it is not your UCI ID number. Be sure your partner has submitted it. Be sure you are providing your and your partner's UCINetID and not your email addresses.*

https://docs.google.com/forms/d/e/1FAIpQLSdjCQeVGIgCdOGepJaJfNb9XScpw0e6DGIiziJ14hXg9DnOg/viewform

Reviewing related material

I encourage you to read your textbook; the section for AVL Trees is clearly marked in the Zybook, and for the Goodrich/Tamassia book's second edition, section 10.1 covers Binary Search Trees and 10.2 talks about AVL Trees. Both books are good at getting to the point, so this should not be a long read. Furthermore, you should look at your notes from the associated lectures.

Requirements

- Fill in MyAVLTree.hpp
    - Your implementation must be implemented via linked nodes in the tree format from the lecture. That is, you may not have a "vector-based tree."
    - Your implementation must fit the interface given.
    - Your implementation must be templated as provided.
        - Be sure yours works for non-numeric types! **char** is a numeric type.
    - You do not need to write a copy constructor or an assignment operator on this project, but knowing how to do so is generally a good thing.
    - You **do** need to implement the destructor.
    - All functions that need to be implemented have been started in the provided .hpp
    - For comparing keys, use the "natural" comparison offered by <.
      Any test cases provided will have something for the key that has this defined.
    - Do not hard code assumptions for how the tree will be used in the second part of the program.
    - Note: the project will not build by default because a reference to a local variable is returned in the `find()` functions. You will need to write an implementation that doesn't do this.

- Write function `void countWords(std::istream & in, MyAVLTree<std::string, unsigned> &counter)` in `proj4.cpp`
    - This function will read words from the input stream. It will break them into given words. All words with the same letters, in the same order, are considered the same word: disregard capitalization. For example, "Bill" and "bill" are the same word, even if the context might imply a person named William in one case and an invoice in the other.
    - You can treat istream just like `std::cin` -- there is some sample code given. Please let me know if it is insufficient, I can explain more.
    - You may assume that the given tree is empty when the function is first called. This will certainly hold true for any test cases.
    - When this function returns, the tree parameter should now have a map of strings to non-negative integers, where the associated value with a given string key is the number of times that word appeared in the stream.
    - It is *strongly recommended* that you produce the tree in the following fashion:

```
for each word in the stream
        if the word has already been seen
                retrieve the count of how many times it has been seen.
                increment that count.
        otherwise
                add this to the tree with a count of 1.
```

You **may not** use parts of the C++ standard template library in this assignment *except* for `std::vector`. Furthermore, `std::vector` may only be used when implementing the three traversals (in-order, pre-order, post-order). For what it's worth, you won't miss it for this assignment. As always, if there's an exception that you think is within the spirit of this assignment, please let me know.

Deliverables

After using the gather script in your project directory to gather up your C++ source and header files into a single **project4.tar.gz** file (as you did in previous assignments[1]), submit that file (and only that file) to Checkmate. Refer back to Project #0 if you need instructions on how to do that. You will need to do this twice: the last code submitted before the checkpoint deadline will be tested for the checkpoint. You will also have to submit for the full project.

You will submit your project via Checkmate. Keep in mind that you're responsible for submitting the version of the project that you want graded. We won't regrade a project simply because you submitted the wrong version accidentally. (It's not a bad idea to look at the contents of your tarball before submitting it; see Project #0 for instructions on how to do that.) *After you submit your checkpoint, I strongly encourage you to rename the project4.tar.gz that you submitted for the checkpoint so you don't accidentally submit it for the full project.*

Can I submit after the deadline?

Yes, it is possible, subject to the late work policy for this course, which is described in the section titled Late work in the course reference and syllabus. **You may not submit for the checkpoint late**.

Grading

Your grade for this project will be graded on correctness: I will run some number of test cases using Google test. Each is worth some number of points and is graded based on whether or not your code correctly determines if the puzzle has a solution, and if so, what it is. If it is determined that your program does not make an attempt to solve the problem at hand, you will not get these points, regardless of the result from testing. The tests will look a lot like the tests in your Google Test starting directory for this assignment; if you pass those, you're off to a good start, but it's not a guarantee. **We will also check for memory leaks; a test case that is correct in result but has memory problems may have reduced (but not zero) credit.**

One sixth of that correctness is the checkpoint, submitted at the date stated. There is no grace period for this portion.

[1] Hopefully you didn't name your previous projects' deliverables "project4.tar.gz" …

The same caveats that applied to previous assignments apply here as well. Make sure your program compiles and that you have tested every function in the AVL Tree! For best results, write Google Tests instead of relying on main. I encourage you to write some test cases before you begin programming the rest of the project.

Your implementation does not have to be the most efficient thing ever, but it cannot be “too slow.” In general, any test case that takes over a minute on the grader’s computer may be deemed a wrong answer, even if it will later return a correct one.