I&C SCI 46 Fall 2021 Project 2: Friends of Friends

Due **October 25, 2021 at 9:59 AM**

Introduction

In lecture, we discussed graphs. In this project, we will begin to explore some related material.

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 in a previous project description, 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 Projects #0 and #1.

Decide on a name for your project directory, then issue the command **ics46 start YOUR_CHOSEN_PROJECT_NAME project2** to create your new project directory using the project1 template. (For example, if you wanted to call your project directory proj2, you would issue the command ics46 start proj2 project2 to create it.) Now you're ready to proceed!

Reviewing related material

I encourage you to read your textbook; in the second edition, section 5.2 deals with queues. Your book is 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. Chapter 3 deals with various ways to implement these, including 3.2 which covers singly-linked lists. The idea of social networks is based on a *graph*, the core data structure we discussed in week three. You are encouraged to read sections 13.1 - 13.3 in the print textbook accordingly as well as re-read the section of the Zybook.

Before asking questions about the requirements or expectations of the second part of this project, be sure to read the Google Test cases provided. I suggest you draw the graphs for the three graph test cases and try to solve them by hand on paper. This will help you figure out how to solve the program.

Requirements

- Fill in LLQueue.hpp
  - Your implementation must be based on the idea of a linked list. This must be a linked list that you implement.
  - Your implementation must fit the interface given
  - Your implementation must be templated
    - An implementation of LLQueue that does not properly template may cause you to get a zero on this assignment. **Do not assume that working for ints is sufficient, despite those being the only cases in the provided test cases.** Test with non-numeric data types as well!
    - **No, really; pay attention to that warning. Students may end up with zeroes for this project if their code is not properly templated, regardless of other considerations.**
  - When attempting to access the front of an empty queue, or to dequeue from an empty queue, you must throw a QueueEmptyException as appropriate.
  - Do not throw an exception upon addition of an element: there is no capacity (beyond what memory allows).
  - **Please note:** the project will not build successfully until you have at least stubbed code for the Queue functions.

  - You also need to write a copy constructor and an assignment operator. These must be “deep copies” -- you should end up with a second queue that has the same contents as the first, not merely two objects each with a pointer to the front of the same linked list.

- Write function `void countPaths(const std::vector< std::vector<unsigned> > & friends, unsigned start, std::vector<unsigned> & pathLengths, std::vector<unsigned> & numShortestPaths)` in proj2.cpp.
    - This problem deals with having a group of people, numbered from 0 to friends.size() - 1. The vector friends[*i*] is a list of who person *i* is friends with. You may assume for this project that friendship is symmetric (if A is friends with B, then B is also friends with A).
    - The last two parameters are provided as initially empty, with size equal to friends.size().
    - Your assignment is to fill these vectors such that:
        - For all *i*, pathLengths[*i*] is the length of the shortest path from *start* to *i* in the graph described by friends.
        - For all *i*, numShortestPaths[*i*] is the number of distinct shortest paths from *start* to *i* in the graph. Figuring out how to compute this is a fundamental part of this assignment.

- 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.

You are prohibited from using any standard library container classes, except for `std::vector` as needed in part two. If you need to use a queue[1], you must use your LLQueue implementation, not `std::queue`. Using a standard library container, including `std::vector`, in implementing your LLQueue in part one will be treated as a serious academic integrity violation.

Deliverables

After *using the gather script* in your project directory to gather up your C++ source and header files into a single **project2.tar.gz** file (as you did in previous assignments), submit that file (and only that file) to Checkmate. Refer back to Project #0 if you need instructions on how to do that. This time, it should give you the correct file name, insert innocent-looking face emoji here.

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.)

[1] Hint: you will want to use a queue in solving the problem in part two.

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.

Grading
The grade that will be sent to you via email is your "raw score" based solely on correctness : we will run some number of test cases for your code and, based on how many and which ones you get correct, you will earn some number of points between 0 and 6 (inclusive, and might not be integer-valued). Each is worth some number of points and is graded based on whether or not your code correctly behaves according to the test case, 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.

The grade that will be emailed does not include late penalties.

If we review your code and find your style to be sufficiently bad, we reserve the right to deduct points based on this, proportional to how bad the style is. If we do so, we will alert you to both the penalty and the reason. Review project 1's style guidelines for an overview.

Academic Integrity

Please review the syllabus for academic integrity expectations for this class. These rules apply even for projects where I don't give you an explicit reminder. As a reminder, the following are **examples** of behavior that is *explicitly prohibited*. This is not a complete list of prohibited behavior:

- Searching online for an implementation of a linked-list based queue
- Looking online for help writing a linked-list (queue or otherwise) in any forum where a potential respondent might not be subject to UCI's academic integrity policy. Of course, if you find help in an allowable resource (such as one of the textbooks for this class), you may use material from them, but it's a good idea to cite this in comments.
- "Hiding" the fact that you are not implementing your own linked list in part one, such as by having a `std::vector` or `std::queue` in your implementation code for LLQueue.
- Looking online for how to find shortest paths in a graph, either the idea or code
    - But you don't have to do this anyway, as the idea should be in your notes by the end of lecture 9, which has already happened before we posted this project, and one of the suggested reading sections of the book has the information too!
- Looking online for help with how to find the number of shortest paths in a graph, especially if you are looking for either the idea or code.