Mechanics of Programming Graph Manipulations

CSCI243 Project 2

10/23/2014

1 Document Version

Revision : 1.4

2 Due Date

Sunday, November 9, by 11:59:59pm.

3 Overview

This project is a study of some of the operations needed for a centralized social network. It is loosely based on the most elementary of features in a system like Facebook®[1] or LinkedIn®.[2] The system is named **AmigoNet**. Your program will maintain a collection of users that have friend ("amigo") connections between them.

The operations you will support are connecting ("friending") and disconnecting ("unfriending") users, as well as determining the distance, or *degrees of separation*, between two users. This kind of data and operations suggests a graph data structure.

4 Assignment

For this assignment, you will implement the functions specified in **amigonet.h**, one of the files you will receive when you execute the `get` command (mentioned in a later section) for this project. After reading this document, look at the documentation contained in that header file for details on how the functions work. Only an overview is given here.

The header file specifies a `UserNode` data structure. That structure contains `Friends`, a pointer to another structure named `Friends_struct`. The definition of that latter `struct` is not defined in **amigonet.h**. Instead you will provide it in **amigonet.c**. This arrangement means that you have the freedom to implement the "edges" of the graph in a wide variety of ways (array, linked list, ...).

---

1. Facebook is a registered trademark of Facebook, Inc.
2. LinkedIn is a registered trademark of LinkedIn Corporation and its affiliates in the United States and other countries.

4.1 Memory Management

To get complete credit, you may not assume a fixed or maximum size for any data item or collection of data. This means that, for example, a list of all users cannot be implemented as an array of predetermined size. Instead, the possibilities include

- Dynamically allocate an array and be prepared to enlarge it when needed.
- Dynamically allocate a hash table and be prepared to enlarge it when needed.
- Use a linked list.

There may be other choices.

Another decision of this type must be made for realizing the "friends" edges, or connections.

It is assumed that a program that tests or otherwise runs your implementation will always begin with a call to `create_amigonet` and end with a call to `destroy_amigonet`. The latter function shall free all memory allocated to AmigoNet.

5 Design Tips

The C language is a lower-level language where you don't have the advantage of a large standard library, and building classic data structures takes a bit more work because of the manual memory management. Think carefully about how to design your collection of users, and users' collections of friends. You are *not* expected to come up with solutions that have optimal space or time complexity; they must just be reasonable.

5.1 Separate the Program into Components

You may create any number of additional C source and header files. It is always a good idea to group code in a cohesive way: Functions and data structures that have a shared purpose should be in the same file, and others with that take care of other concerns should be in separate files.

If you create what you consider "private" or internal functions to any C source file you write, including `amigonet.c`, it is preferable to label them `static` to avoid name clashes with other code that may be linked in with yours. Do the same for global data structures, as they should not be shared outside a single C file's code.

5.2 The separation Function

You may end up spending half your time on the function `separation`. It involves a graph traversal to find the shortest distance between two nodes (users) in the graph.

The standard approach is to do a breadth-first search (*BFS*) starting at the source node and stopping when you reach the destination node. You are certainly allowed to do it this way, but keep in mind the following.

- You will need a queue data structure for the BFS algorithm.
- Each node you insert in your queue will have to have associated with it its distance from the source.
- You will need a "visited" set to store nodes you have already seen. (Recall this avoids infinite cycles.)

An alternative that is a bit less work is to use *Deepening Depth-First Search.* It iteratively performs depth-first searches that stop when a certain depth (distance from source) is reached, incrementing the maximum depth each time. The algorithm returns the maximum depth used when the destination is found for the first time. The search must be exhaustive, e.g., no "visited" set, to ensure that all paths are tried.

A big problem with Deepening DFS is when two nodes are not connected. What is the maximum depth you must try before you give up? If that maximum is high, the separation function could take a while, even a few minutes, to run, depending on other factors. If you choose this approach, you may want to first check if the two nodes are connected by implementing a standard DFS, with a "visited" list. Then you don't need a maximum depth because you know some depth value will succeed.

6 Testing

To help you develop tests more easily, a test program named `amigosim.c` is provided that reads AmigoNet commands from standard input. A simple beginner's input file is also provided, along with the output your program should produce if it reads the file. There is documentation in `amigosim.c` regarding the format of commands. You are encouraged to create your own input files test your program. You also may want to write your own test programs that don't rely on input, but do strange things in the sequence of calls into **amigonet**.

7 Supplied Files

There are some files that you will use when building and testing your code. Retrieve them using this command:

```
get  csci243  project2
```

This will create the following files in your working directory:

- **amigonet.h** - The interface for the functions you will write
- **amigosim.c** - The test program that takes amigonet commands from standard input
- **scenario1.txt** - A small input file for **amigosim**
- **scenario1.out** - The correct output for **scenario1.txt**

You should not modify these files!

In particular, don't make any changes to the **amigonet.h** header file, because you will not be submitting it and our test programs will be using the version we have supplied to you.

8 Submission

You will need to create and submit at least one C source file for this assignment:

- **amigonet.c**

As usual you are most welcome to submit a `README.txt` file to clarify how things work.

The submission command is

```
try grd-243 project2-1 amigonet.c  other-files
```

where *other-files* is any additional `.c` and/or `.h` files you are submitting, or `README.txt`, or a `revisions.txt` file if a revision log is not put in your source files by your versioning system.

Note that you are *not* submitting a main program (file with a `main` function in it)!

9 Grading

Your program will be graded based on the following criteria for a total of 80 points.

- (40 pts) Functionality, split among the functions you were to write for AmigoNet
  - (4 pts) `create_amigonet`
  - (6 pts) `destroy_amigonet`
  - (6 pts) `addUser`
  - (6 pts) `findUser`
  - (4 pts) `addAmigo`
  - (4 pts) `removeAmigo`
  - (8 pts) `separation`
  - (2 pts) `dump_data`
- (10 pts) Reasonableness of data structures and their algorithms
- (10 pts) Quality of design. This includes things like the splitting of the code into separate files to support cohesiveness and separation of concerns, using `static` for private utility functions and data, and avoiding the duplication of code.
- (10 pts) Programming Style: Are you following a consistent and reasonable coding style? Is your code reasonably structured?
- (10 pts) Documentation: Is your code properly commented and documented? Please check with your individual instructor regarding any additional requirements (e.g. version control).

A program that cannot be compiled and linked by `try` will not be accepted, and will receive a grade of '0'.