Degrees of LinkedIn

Due: Monday June 20 @ 10am pushed to your GitHub Repo for PA03

Introduction

When you log in to LinkedIn and search for a profile of someone with whom you are not connected, you might see info related to mutual connections. This can be useful when trying to get introduced to someone that you don't already know. Having a mutual connection make an introduction for you can be more productive than just a cold email. Or perhaps you've heard of 6 Degrees of Separation, which suggests that everyone on earth is connected by no more than 5 intermediate people. Determining a chain of connections is a classical computer science problem.

In this project, you'll implement an algorithm and associated data structures to find the shortest chain of connections possible between two people.

For example, assume the following social network from LinkedIn:

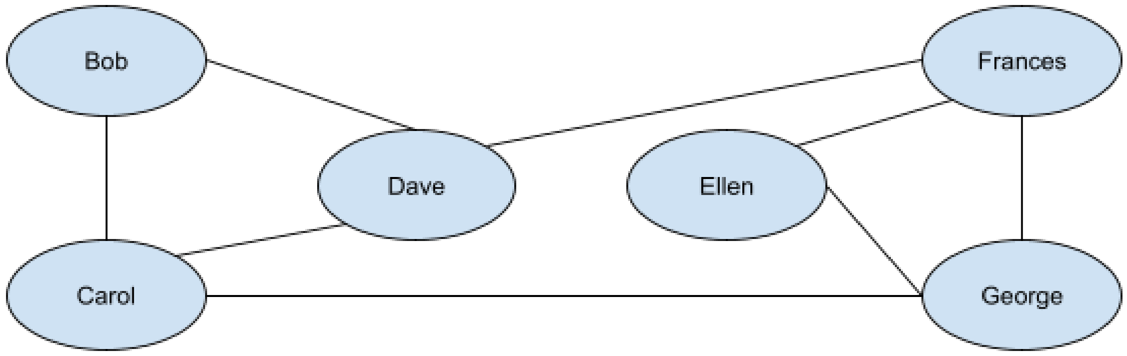

Figure 1 - a graphical representation of connected users on LinkedIn.

Each oval represents an individual; each line connecting two ovals represents two individuals being linked. We call the line connecting two individuals an **edge.**

In this project, you'll determine some characteristics of the social graph as well as characteristics of connections between certain individuals.

Input File

The program will accept 1 input file that will have the set of connected individuals as well as the set of shortest linked path requests. A linked path request is composed of the names of two individuals and your task is to algorithmically determine the shortest path of connections between those two people.

Connections & Requests Data

Here is an example of a data input file (it is not meant to match the figure above):

```
4
Bob Smith - Sally Simpson
Sally Simpson - Daniel Davidson
Sally Simpson - Susan Jackson
Susan Jackson - Daniel Davidson
3
Bob Smith - Daniel Davidson
Sally Simpson - Daniel Davidson
Bob Smith - Bob Smith
```

The input file will contain a set of connections followed by a set of requests.

- The first line of the file will contain an integer *n* indicating how many connections are in the file. Each of the next *n* rows contain two individuals' names separated with a hyphen. Each name will be made up of a first name and last name separated by a space. No first or last names will have spaces in them.
- The next line of the file contains an integer *m* indicating the number of "degrees of separations" requests. The next *m* lines contain two names per line separated by a hyphen. No name will be requested that doesn't exist in the connections part of the file.

Output File

Output will be written to a file. The name of the file will be a command line argument.

For each shortest path request from the input file (the bottom part of the file), output:

- the request as stated in the input file followed by a colon and new line
- the path that contains the smallest number of people connecting the two individuals from the request. Each person's name should be separated with a right-pointing angle bracket ('>'). If there are multiple paths of the same length, output all of them in no particular order.

Data Structure Implementation

The DSLinkedList Class

- The DSLinkedList class shall be an implementation of a doubly linked list.
- A separate class to represent the node shall also be included.
- Both classes should be templated so that a doubly linked list can support any kind of data payload.
- Pay special attention to dynamic memory management and include method implementations consistent with Rule of 3.

You are responsible for determining the interface of your LinkedList class. Your LinkedList class must include some mechanism to allow other code (client code) to iterate over the list without exposing the underlying implementation. What this means is that a programmer using your LinkedList class should be able to step through the sequence of elements contained within (minimally, from front of list to end of list) without concern for how such iterating is accomplished (you may consider implementing an entire iterator class, or something similar).

The Adjacency List

In order to store the structure representing connections between individuals (a social network), you will implement a simple adjacency list data structure. Essentially, it will be a linked list of linked lists. There will be one linked list for every distinct individual. Each list will contain the individuals that are connected to a given individual. **Figure 2** is an example representation of an adjacency list for the graph in **Figure 1**.

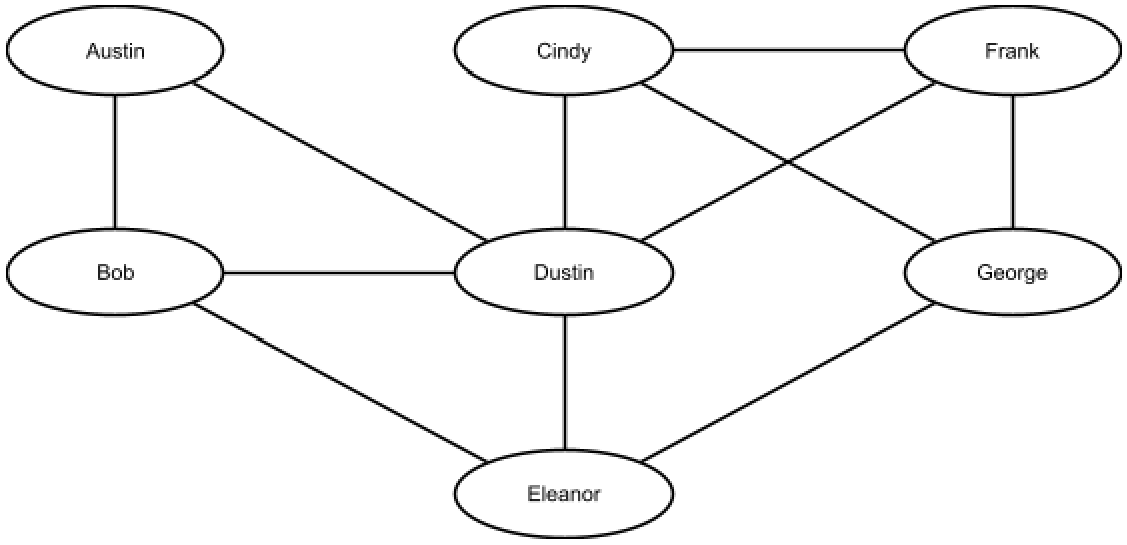

Figure 2: A graph representing connections between users.

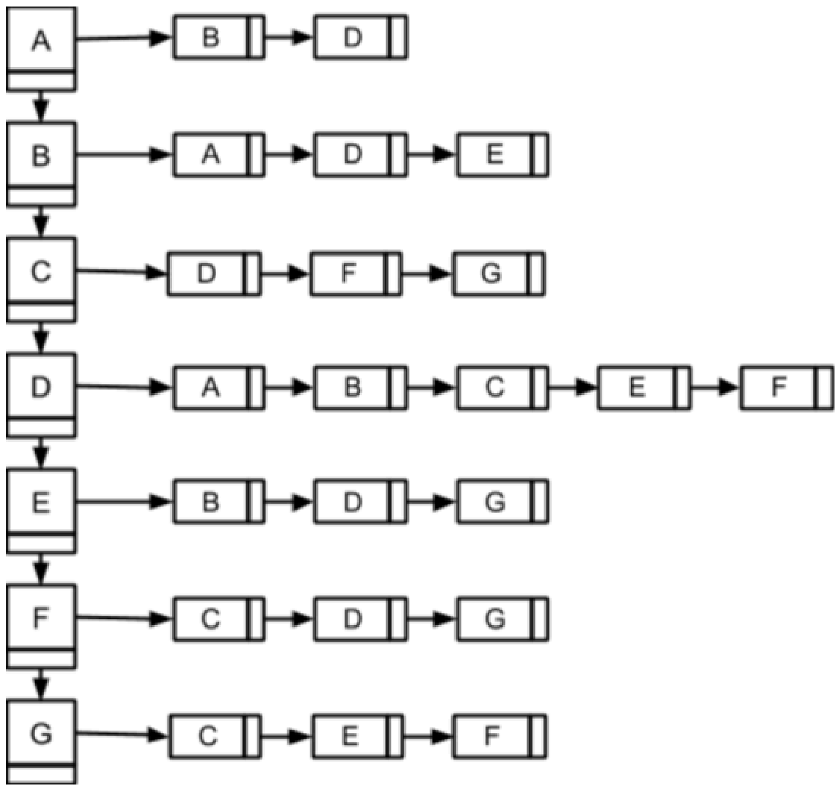

Figure 3: An **adjacency list** representation of the graph from figure 2.

The larger squares on the left represent the container of individuals (with one node for each individual). The list to which each node is pointing represents the individuals to whom the individual in the head of said list is connected to. For example, Person A (Austin) is connected to Person B (Bob) and Person D (Dustin).

Finding the Shortest Path

Finding the shortest path between two nodes in a graph is not a trivial process. While there are more advanced algorithms, the process we are going to use will generate all the paths and then find the one with the least number of nodes.

To enumerate all the paths from one user to another, you'll implement an algorithm called **iterative backtracking**. We will discuss iterative backtracking in class.

Some Implementation Requirements:

- For this project, you may NOT use any of the STL container classes or associated algorithms except for **std::stack<T>**.
- You need to implement Catch2 tests for your DSLinkedList, but not for the DSAdjacencyList class.
- Your implementation should be object-oriented in both design and implementation. Minimize the amount of code you have in your main method.

Executing Your Program

The final version of your program will be run from the command line with the following arguments:

```
./linkedin <dataInputFile> <OutputFile>
```

Grading

Your project will be graded by one of the TAs for the course.

Outcome	Points	Points Earned
DSLinkedList and Tests	30	
Adjacency List	15	
Iterative backtracking implementation	20	
Source code quality	15	
Full functionality	20	