Maxwell “Lord Vader” Bernstein, Erica Schwartz

Assigned: Thursday, April 7th, 2016
Due: Thursday, April 21st, 2016

COMP 15 Homework 6-Degrees

1. Prelude

You’re a COMP 105 student. You notice that you were your current TA’s COMP 15 TA back in the day. You wonder, “How many ‘TA-ing degrees of separation’ are there between me and my friends?” You think back on your data structures training from COMP 15 and notice that this project is a graph problem.

After begging the department administrators to give you rosters and TAs of every class for the past seven years, you sit down to write your program...

2. Overview

You will read in files containing TA lists and course rosters going many semesters back, and store this input in some data structure(s). You will store information about which students have TAed which other students. (We say student A has TAed student B if student A was a TA for a given course during the same semester that student B took that course.)

Then, you will print a prompt, >>, and read one of the following commands from standard input (cin):

1. `ls`
   Lists the students found in both of the inputted files. Student names should be terminated by newlines.
2. `lc`
   Lists the courses found in both of the input files. Course names should be terminated by newlines.
3. `taed <student_name>`
   Lists the courses that `<student_name>` has TAed. Course names should be terminated by newlines.
4. `roster <course name>`
   Lists the roster of the class `<course_name>`. Student names should be terminated by newlines.
5. `paths <student_a> <student_b>`

Print all of the paths between `<student_a>` and `<student_b>`. Each path should be formatted as above. Paths should be terminated by newlines. A path cannot contain the same student more than once. You may want to use a breadth-first search to find these paths.

6. `shortestpath <student_a> <student_b>`
   Find the shortest path between `<student_a>` and `<student_b>`. If multiple paths have the shortest length, output any one. Paths should be output in the following format:

```
Brad.Pitt +- 15S13 -> George.Clooney +- 11F14 -> Tom.Cruise
```

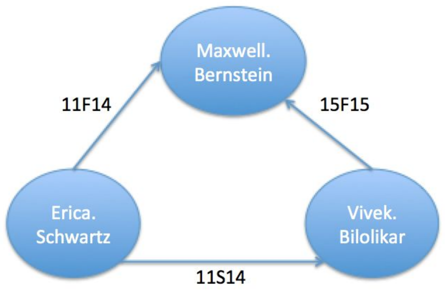
A typical session with `hw6degrees` might look like:

```
>> ls
Brad.Pitt
George.Clooney
Tom.Cruise
>> lc
15S13
11F14
>>
```

3. Examples

Part A

Say Erica.Schwartz TAed Vivek.Bilolikar in comp11 in spring 2014 and Maxwell.Bernstein in comp11 in fall 2014, and then Vivek.Bilolikar TAed Maxwell.Bernstein in comp15 in fall 2015. This describes the following graph:



Your input will come in the form of two input files. In this case, the first file would look like this (Fig. 1):

```
Maxwell.Bernstein:11:F14
Maxwell.Bernstein:15:F15
Vivek.Bilolikar:11:S14
```

And the second file would look like this (Fig. 2):

```
Erica.Schwartz:11:F14
Erica.Schwartz:11:S14
Vivek.Bilolikar:15:F15
```

You will read in and store this information, and then wait for input from stdin.

When the user enters:

```
ls
```

They are asking for a list of students, and you should print:

```
Erica.Schwartz
Maxwell.Bernstein
Vivek.Bilolikar
```

When the user enters:

```
lc
```

They are asking for a list of courses, and you should print:

```
11F14
11S14
15F15
```

When the user enters:

```
taed Vivek.Bilolikar
```

They are asking for a list of courses that Vivek.Bilolikar has TAed, and you should print:

```
15F15
```

When the user enters:

```
roster 11F14
```

They are asking for the roster of comp11 in fall 2014 and you should print:

```
Maxwell.Bernstein
```

When a user enters:

```
paths Erica.Schwartz Maxwell.Bernstein
```

They are asking for paths from Erica.Schwartz to Maxwell.Bernstein, and you should print:

```
Erica.Schwartz +- 11F14 -> Maxwell.Bernstein
Erica.Schwartz +- 11S14 -> Vivek.Bilolikar +- 15F15 ->
Maxwell.Bernstein
```

When the user enters:

```
shortestpath Vivek.Bilolikar Maxwell.Bernstein
```

They are asking for the shortest path from Vivek.Bilolikar to Maxwell.Bernstein, and you should print:

```
Vivek.Bilolikar +- 15F15 -> Maxwell.Bernstein
```

Part B

Now, consider the following graph:

In this case, the first file would look like this:

```
George.Clooney:15:S13
Jennifer.Lawrence:15:F15
Kevin.Bacon:15:F15
Meryl.Streep:40:S14
Tom.Cruise:11:F14
Tom.Cruise:160:S16
Tom.Cruise:105:S16
```

And the second file would look

like this:

```
Brad.Pitt:15:S13
George.Clooney:11:F14
George.Clooney:40:S14
Jennifer.Lawrence:160:S16
Kevin.Bacon:105:S16
Meryl.Streep:15:F15
```

When the user enters:

```
paths Brad.Pitt Tom.Cruise
```

You should print:

```
Brad.Pitt +- 15S13 -> George.Clooney +- 11F14 -> Tom.Cruise
Brad.Pitt +- 15S13 -> George.Clooney +- 40S14 -> Meryl.Streep +- 15F15
  -> Jennifer.Lawrence +- 160S16 -> Tom.Cruise
Brad.Pitt +- 15S13 -> George.Clooney +- 40S14 -> Meryl.Streep +- 15F15
  -> Kevin.Bacon +- 105S16 -> Tom.Cruise
```

When the user enters:

```
paths George.Clooney Kevin.Bacon
```

You should print:

```
George.Clooney +- 40S14 -> Meryl.Streep +- 15F15 -> Kevin.Bacon
```

When the user enters:

```
paths Tom.Cruise Brad.Pitt
```

Your program should not produce any output.

When the user enters:

```
paths Oprah.Winfrey Tom.Cruise
```

or:

```
paths George.Clooney Jimmy.Fallon
```

or:

```
paths Oprah.Winfrey Jimmy.Fallon
```

You should print:

```
Student not found.
```

4. Implementation Specifics

Your program should be runnable by typing:

```
make
./hw6degrees students.txt tas.txt
```

into the terminal, where `students.txt` is a file in the directory that contains input as formatted in Fig. 1 and `tas.txt` is a file in the directory that contains input as formatted in Fig. 2.

You will store students and courses in data structure(s) of your choice. You are welcome to use the C++ Standard Template Library for queues, stacks, vectors, and/or sets, but you must implement all other data structures yourself.

We provide you with a hash function (in hashfunc.cpp and hashfunc.h), should you choose to implement a hash table. We also provide you with a Makefile that you will need to edit when you add .h and .cpp files of your own. Copy these files we're providing by using the following command:

```
mkdir hw6; cd hw6
cp /comp/15/files/hw6/* ./
```

5. Implementation Plan

As in HW5, this project will be split into steps. You should be continually testing and debugging your code along the way. Be careful to keep any possible edge cases in mind.

The stages are defined roughly as follows:

Stage zero

Think. Don't open an editor right away. What is being asked of you? Review the pictures and examples above. Do you understand what information is being represented? Can you find paths by hand? Once you're there, make a list: what information do you need to store, i. e., what is the data? Just brainstorm at first. You can do this with others (you can't share code, but you help each other understand the problem). For different kinds of data, what will you need to do with it? You may start thinking of some classes if some data elements have lots of state or operations. Make a list of such things. We started this in class already. Use paper. But you can also start putting things in your ReadMe file, which you can update as you go.

Stage one

Design your solution — choose your data structures, and outline your classes, your .h files, and your call tree. Aim to be finished with this stage by Monday, April 11th.

Stage two

Now you can start to implement your data structures, *and* you can work on the program skeleton. You can do this in different orders. If you have classes, then you can implement them and test them with specialized main functions/files.

For the assigned main program, you need to get some basic things going: Read in Student and TA files. Start by just reading each file in and printing it. Then, when you have some of your data structures working and you have the basic input working, read in the files and store them in your data structures. Now you can implement the user interaction commands `ls` and `lc`. At this point, you can be pretty sure you are correctly reading in, storing, and retrieving the essential data. If you can't do these things, you can't debug the rest of the project, so it's important to get this done.

Aim to be finished with this stage by Friday, April 15th.

Stage three

Implement `taed`, `roster`, `paths` and `shortestpath` commands, as well as all other unfinished aspects of the project.

HW6 will be due on Thursday, April 21st.

Stage four

Turn it in. When you provide your work, do not forget to update the ReadMe and Makefiles. You will have to add your files to the provide command in the Makefile, too.

Congratulations!