CSCI 2600 — Principles of Software

Homework 5: The Marvel Comics Universe

Due: Monday, April 10, 2023, 11:59:59 pm

Submission Instructions

- Follow the directions in the version control handout for cloning your hw04 git repo. The URI that you will need to use to clone your personal repo for this homework would have the form of `https://submitty.cs.rpi.edu/git/s23/csci2600/hw04/RCSID` where RCSID is your RCS ID.
- Be sure to commit and push the files to Submitty. Follow the directions in the version control handout for adding and committing files.
- Be sure to add any additional files to your repo using Team/Add to Index.
- **Important:** You must press the **Grade My Repository** button, or your answers will not be graded.

IMPORTANT NOTES:

You should have package `hw5` with the usual directory structure. Write your code under `src/main/java/hw5` and your tests under `src/test/java/hw5` (shows as `hw5` under `src/test/java` in Package Explorer). If your directory structure is incorrect autograding will fail resulting in a grade of 0!

Introduction

In this homework, you will put the graph you designed in Homework 4 to use by modeling the Marvel Comics universe. By trying out your ADT in a client application, you will be able to test the usability of your design as well as the correctness, efficiency, and scalability of your implementation. You may have to overhaul, or just tweak your implementation, once you try it with a larger load.

The application builds a graph containing thousands of nodes and edges. At this size, you may discover performance issues that weren't revealed by your unit tests on smaller graphs. With a well-designed implementation, your program will run in a matter of seconds. Bugs or less ideal choices of data structures can increase the runtime to anywhere from several minutes to 30 minutes or more. If this is the case you may want to go back and revisit your graph implementation from Homework 4. Remember that different graph representations have widely varying time complexities for various operations and this, not a coding bug, may explain the slowness.

The MarvelPaths Application

In this application, your graph models a social network among characters in Marvel comic books. Each node in the graph represents a character, and an edge ⟨`Char1,Char2`⟩ indicates that Char1 appeared in a comic book that Char2 also appeared in. There should be a separate edge for every comic book, labeled with the name of the book. For example, if Zeus and Hercules appeared in five different issues of a given series, then Zeus would have five edges to Hercules, and Hercules would have five edges to Zeus.

Your graph should not store reflexive edges from characters to themselves.

You will write a class `hw5.MarvelPaths` (in file `MarvelPaths.java` in package `hw5`) that reads the Marvel data from a file (`marvel.csv`), builds a graph, and finds paths between characters in the graph. You are not required to write a main method as a driver for your application; nevertheless, we encourage you to do so, both for your own convenience in testing and for the satisfaction that comes with creating a complete, standalone application. Do not implement the graph in `MarvelPaths`. `MarvelPaths` should use your graph class from `hw4` by composition.

As you complete this assignment, you may need to modify the implementation and perhaps the public interface of your Graph ADT from Homework 4. Briefly document any changes you made and why in `answers/hw5_changes.pdf` (no more than 1-2 sentences per change). If you made no changes, state that explicitly. You don't need to track and document cosmetic and other minor changes, such as renaming a variable; we are interested in substantial changes to your API or implementation, such as adding a public method or using a different data structure. Describe logical changes rather than precisely how each line of your code was altered. For example, "I switched the data structure for storing all the nodes from a ___ to a ___ because ___" is more helpful than "I changed line 27 from `nodes = new ___();` to `nodes = new ____();`".

Leave your graph in the `hw4` package where it was originally written, even if you modify it for this assignment. There is no need to copy files or duplicate code! You can just `import hw4` and use it in Homework 5. If you do modify your `hw4` code, be sure to commit your changes to your repository.

Do not make `MarvelPaths` a subclass of your `Graph` class. Instead, use `Graph` in `MarvelPaths` by composition.

Problem 1: Getting the Marvel Universe Data

Before you get started, obtain the Marvel Universe dataset. We have not added this file to your Git repositories because it is fairly large. Instead, download the file from the course Web site. Store the file in `data/marvel.csv`. (You might have to add the data directory under the project root.)

IMPORTANT: Do not commit the marvel.csv into your repository! There is a limit on each repository and committing such a large file may break this limit. This is easily taken care of in eGit, you can simply exclude marvel.csv.

Take a moment to inspect the file. A CSV (”comma-separated value”) file consists of human-readable data delineated by commas, and can be opened in any advanced text editor, like Notepad++. We do not recommend using basic text editors like standard Microsoft Windows Notepad, a spreadsheet like Microsoft Excel, or Eclipse to view or edit CSV files because such tools may not show the contents of CSV files correctly or may make undesired changes to the files causing parsing errors when your code attempts to read the data. Each line in `marvel.csv` is of the form:

"character","book"

where character is the name of a character, book is the title of a comic book that the character appeared in, and the two fields are separated by a comma. Note that while there exist several different variations of the CSV format, the one used in this assignment assumes that every value must be enclosed in double quotes (the ” character). These double quotes only serve as delimiters, i.e., they are not part of the actual character name or book title.

Problem 2: Building the Graph [27 points with Problem 3]

The first step in your program is to construct your graph of the Marvel universe from a data file. We have written a class `MarvelParser` that may help. `MarvelParser` has one static method, `readData()`, which reads data from `marvel.csv`, or any file structured in the same format. `readData()` creates in-memory data structures: a `Set` of all characters and a `Map` from each book to a `Set` of characters in that book. These are not the data structures you want, however; you want a `Graph`.

We have also included a main method which takes the file name as a command-line argument and then calls `readData()`. To add `marvel.csv` as a command-line argument in Eclipse, go to Run->Run Configurations. In the Run Configurations window, choose the ”Arguments” tab. In ”Program Arguments”, type `data/marvel.csv`.

Later on when measuring coverage, you can comment out the main method. (Since it is never called from the tests, it will decrease your percent coverage.) If you choose not to use our parser code, you can get rid of the file altogether. You do need to substitute with your own parser code though.

You may modify `MarvelParser` however you wish to fit your implementation. You may change the method signature (parameters and return value) of `parseData()`, or you may leave `parseData()` as is and write code that processes its output. The only constraint is that your code needs to take a filename as a parameter so the parser can be reused with any input file.

At this point, it's a good idea to test the parsing and graph-building operation in isolation. Verify that your program builds the graph correctly before you go on. The assignment formally requires this in Problem 5.

Problem 3: Finding Paths [27 points with Problem 3]

The real meat (or tofu) of `MarvelPaths` is the ability to find paths between two characters in the graph. Given the name of two characters, `MarvelPaths` searches for and returns a path through the graph connecting them. How the path is subsequently used, or the format in which it is printed out, depends on the requirements of the particular application using `MarvelPaths`.

Your program should return the shortest path found via breadth-first search (BFS). A BFS from node u to node v visits all of u's neighbors first, then all of u's neighbors' neighbors, then all of u's neighbors' neighbors' neighbors, and so on until v is found or all nodes with a path from u have been visited. Below is a general BFS pseudocode algorithm to find the shortest path between two nodes in a graph G. For readability, you should use more descriptive variable names in your actual code than are needed in the pseudocode:

```
start = starting node
dest = destination node
Q = queue, or "worklist", of nodes to visit: initially empty
M = map from nodes to paths: initially empty.
    // Each key in M is a visited node.
    // Each value is a path from start to that node.
    // A path is a list; you decide whether it is a list of nodes, or edges,
    // or node data, or edge data, or nodes and edges, or something else.

Add start to Q
Add start->[] to M (start mapped to an empty list)
while Q is not empty:
    dequeue next node n
    if n is dest
        return the path associated with n in M
    for each edge e=⟨n,m⟩:
        if m is not in M, i.e., m has not been visited:
            let p be the path n maps to in M
            let p' be the path formed by appending e to p
            add m->p' to M
            add m to Q

If the loop terminates, then no path exists from start to dest.
The implementation should indicate this to the client.
```

Here are some facts about the algorithm.

- It is a loop invariant that every element of Q is a key in M
- If the graph were not a multigraph, the for loop could have been equivalently expressed as for each neighbor m of n:
- If a path exists from start to dest, then the algorithm returns a shortest path.

Many character pairs will have multiple paths. **For grading purposes, your program should return the lexicographically (alphabetically) least path.** More precisely, it should pick the lexicographically first character at each next step in the path, and if those characters appear in several comic books together, it should print the lexicographically lowest title of a comic book that they both appear in. The BFS algorithm above can be easily modified to support this ordering: in the for-each loop, visit edges in increasing order of m's character name, with edges to the same character visited in increasing order of comic book title. This is not meant to imply that your graph should store data in this order; it is merely a convenience for grading.

Because of this application-specific behavior, **you should implement your BFS algorithm in** `MarvelPaths` rather than directly in your graph, as other hypothetical applications that might need BFS probably would not need this special ordering. Further, other applications using the graph ADT might need to use a different search algorithm, so we don't want to hard-code a particular search algorithm in the graph ADT.

Using the full Marvel dataset, your program must be able to construct the graph and find a path in just a couple of seconds on your PC/laptop and on Submitty. When running tests we will set a 10 second timeout for each test suite. Note that if your solution exceeds the limit, your process is terminated and then the output file is truncated and you may receive a strange error message about incorrect formatting of the output file.

Similarly to Homework 4, add an instance field that stores a Graph in `MarvelPaths`. For testing purposes, we require that you implement the following public methods in `MarvelPaths`. Otherwise, design class `MarvelPaths` as you wish and add operations as you wish.

```
public void createNewGraph(String filename)
```

The method creates a brand new graph in the instance field in `MarvelPaths` and populates the graph from *filename*, where *filename* is a data file of the format defined for`marvel.csv` and is located in the data/ directory of your project.

```
public String findPath(String node1, String node2)
```

Find the shortest path from *node1* to *node2* in the graph using your breadth-first search algorithm.

Paths should be chosen using the lexicographic ordering described above. If a path is found, the returned String should contain the path in the format below. That is,
`System.out.print(mp.findPath("CHAR`$_1$`","CHAR`$_N$`"))` where `mp` refers to an instance of `MarvelPaths`, prints the following:

`path from CHAR`$_1$` to CHAR`$_N$`:`
`CHAR`$_1$` to CHAR`$_2$` via BOOK`$_1$
`CHAR`$_2$` to CHAR`$_3$` via BOOK`$_2$
`...`
`CHAR`$_{N-1}$` to CHAR`$_N$` via BOOK`$_{N-1}$

where $CHAR_1$ is the first node listed in the arguments to findPath, $CHAR_N$ is the second node listed in the arguments of findPath, and $BOOK_K$ is the title of a book that CHAR$_K$ and CHAR$_K$+1 appeared in.

For example, `mp.findPath("Zena","Zeus")` will construct and return the String:

```
"path from Zena to Zeus:\nZena to Hercules via Book1\nHercules to Zeus via Book2\n"
```

(We are just making this up to illustrate the placement of newline characters, there is no such a path in the Marvel universe.)

Not all characters may have a path between them. If the user gives two valid node arguments that have no path in the specified graph, output the following as String:

```
path from CHAR 1 to CHAR N:
no path found
```

If a character name *CHAR* was not in the original dataset, simply output:

```
unknown character CHAR
```

If neither character is in the original dataset, output the line twice: first for the first node, then for the second one. These should be the only lines your program produces in this case — i.e., do not output the regular "path from ..." or "path not found" output.

What if the user asks for the path from a character in the dataset to itself? A trivially empty path is different from no path at all, so the "no path found" output isn't appropriate here. But there are no edges to print, either. So you should output the header line

```
path from C to C:
```

but nothing else. (Hint: a well-written solution requires no special handling of this case.)

This only applies to characters in the dataset: a request for a path from a character that is not in the dataset to itself should have the usual "unknown character *C*" output.

In all cases the string should end with \n (newline), just like in the first case.

The following examples illustrate the required format of the output. Among multiple paths of length 5 between ”PETERS, SHANA TOC” and ”SEERESS”, `findPath()` needs to select the lexicographically least path which would be the first of the three paths listed below:

```
path from PETERS, SHANA TOC to SEERESS:
PETERS, SHANA TOC to KNIGHT, MISTY via M/CP 80/3
KNIGHT, MISTY to ALEXANDER, CALEB via N 17
ALEXANDER, CALEB to HULK/DR. ROBERT BRUC via N@ 1/3
HULK/DR. ROBERT BRUC to RAVAGE/PROF. GEOFFRE via RH2 2
RAVAGE/PROF. GEOFFRE to SEERESS via M/CP 117/4

path from PETERS, SHANA TOC to SEERESS:
PETERS, SHANA TOC to KNIGHT, MISTY via M/CP 80/3
KNIGHT, MISTY to ALEXANDER, CALEB via N 19
ALEXANDER, CALEB to HULK/DR. ROBERT BRUC via N@ 1/3
HULK/DR. ROBERT BRUC to RAVAGE/PROF. GEOFFRE via RH2 2
RAVAGE/PROF. GEOFFRE to SEERESS via M/CP 117/4

path from PETERS, SHANA TOC to SEERESS:
PETERS, SHANA TOC to WING, COLLEEN via M/CP 80/3
WING, COLLEEN to ALEXANDER, CALEB via N 8
ALEXANDER, CALEB to HULK/DR. ROBERT BRUC via N@ 1/3
HULK/DR. ROBERT BRUC to RAVAGE/PROF. GEOFFRE via RH2 2
RAVAGE/PROF. GEOFFRE to SEERESS via M/CP 117/4
```

To help you with formatting your output correctly, we provide several sample files described below:

Description	**An example of the call to** `findPath()`	**Sample file**
A path is found. The lexicographically (alphabetically) least path is returned.	`System.out.print (mp.findPath("PETERS, SHANA TOC", "SEERESS"));`	sample_hw5_output_00.txt
No path exists.	`System.out.print (mp.findPath("GOOM", "HOFFMAN"));`	sample_hw5_output_01.txt
Character not found.	`System.out.print (mp.findPath("BATMAN", "CAPTAIN AMERICA"));`	sample_hw5_output_02.txt
Both characters not found.	`System.out.print (mp.findPath("BATMAN", "GREEN LANTERN"));`	sample_hw5_output_03.txt
A path to the character themselves.	`System.out.print (mp.findPath("SEERESS", "SEERESS"));`	sample_hw5_output_04.txt
A path to the unknown character themselves.	System.out.print (mp.findPath(”BATMAN”, ”BATMAN”));	sample_hw5_output_05.txt

Problem 4: Testing Your Solution

Because the Marvel graph contains literally thousands of nodes and hundreds of thousands of edges, using it for correctness testing is probably a bad idea. By contrast, using it for scalability testing is a great idea, but should come after correctness testing has been completed using much smaller graphs. In addition, it is important to be able to test your parsing/graph-building and BFS operations in isolation, separately from each other.

You should first write `*.csv` files in the format defined for `marvel.csv` to test your `MarvelPaths`. All these files will go in the `data/` directory.

Write tests in the regular JUnit test class in folder `src/test/java/hw5` (use `hw5` package). You might have to create the folder `src/test/java/hw5` and specify that you want to put your test class in the `hw5` package. Make sure that you handle the edge cases. You will have to specify data files to load in your implementation tests, **so make sure you read the File Paths section for information about specifying filenames very carefully.**

As in Homework 4, run EclEmma and measure coverage of your tests. We will be measuring coverage too. The code coverage threshold will be set at 80% for this assignment.

Reflection [0.5 points]

Please answer the following questions in a file named `hw5_reflection.pdf` in your `answers/` directory. Answer briefly, but in enough detail to help you improve your own practice via introspection and to enable the course staff to improve Principles of Software in the future.

(1) In retrospect, what could you have done better to reduce the time you spent solving this assignment?

(2) What could the Principles of Software staff have done better to improve your learning experience in this assignment?

(3) What do you know now that you wish you had known before beginning the assignment?

Collaboration[0.5 points]

Please answer the following questions in a file named `hw5_collaboration.pdf` in your `answers/` directory.

The standard integrity policy applies to this assignment.

State whether you collaborated with other students. If you did collaborate with other students, state their names and a brief description of how you collaborated.

Grade Breakdown

- Quality of test suite, percent of your tests passed: 6 pts (auto-graded)
- Quality of test suite, percent coverage: 6 pts (auto-graded)
- Instructor MarvelPaths small tests: 7 pts (auto-graded)
- Instructor MarvelPaths large tests: 8 pts (auto-graded)
- Changes (`answers/hw5_changes.pdf`): 10 pts
- Code quality (`hw5/*.java`, specs, rep invariants, AFs, etc.): 12 pts
- Collaboration and reflection (`answers/hw5_reflection.pdf` and `answers/hw5_collaboration.pdf`): 1 pt

Paths to Files

When you use test files in `data/`, hardcode the relative path in your tests. For example, if you use file `testfile1.csv` in directory `data/`, you can load the file using `BufferedReader reader = new BufferedReader(new FileReader("data/testfile1.csv"))`.

Behavior may vary from one version of Eclipse to another. As long as you hardcode your relative paths starting at `data/` you will be fine on Submitty.

Hints

Performance

If your program takes an excessively long time to construct the graph for the full Marvel dataset, first make sure that it parses the data and builds the graph correctly for a very small dataset. If it does, here are a few questions to consider:

- What data structures are you using in your graph? What is their ”big-O” runtime? Are there others that are better suited to the purpose?
- Did you remember to correctly override `hashCode()` if you overrode `equals()`?
- What is the ”big-O” runtime of your `checkRep()` function? Does performance improve if you comment it out?

Miscellaneous

As always, remember to:

- Use descriptive variables names (especially in the BFS algorithm) and inline comments as appropriate.
- Include an abstraction function, representation invariant, and `checkRep` in all classes that represent an ADT. If a class does not represent an ADT, place a comment that explicitly says so where the AF and RI would normally go. (For example, classes that contain only static methods and are never constructed usually do not represent an ADT.) Please come to office hours if you feel unsure about what counts as an ADT and what doesn't.

What to Submit

You should add and commit the following files to your hw04 Git repository:

- `src/main/java/hw5/MarvelPaths.java`
- `src/main/java/hw5/*.java` *[Other classes you create, if any (there may be none!)]*
- `answers/hw5_reflection.pdf`
- `answers/hw5_collaboration.pdf`
- `data/*.csv` (excluding `marvel.csv`) [Your .csv test files. Don't commit marvel.csv. Right click this file in your `Git Staging` panel of Eclipse and select `Ignore`.]
- `src/test/java/hw5/*Test.java` *[JUnit test classes you create.]*

Additionally, be sure to commit any updates you may have made to the following files, so the staff has the correct version for this assignment:

- `src/main/java/hw4/*.java` *[Your graph ADT.]*
- `src/main/java/hw5/MarvelParser.java`

Errata

None yet. Check the Submitty Discussion forum regularly.

Parts of this homework were copied from the University of Washington Software Design and Implementation class by Michael Ernst.