Assignment Two (A2): Research Network! Recursion, Binary Search Trees, and Graphs

Troy Astarte & Daniel Archambault

2022-03-21

Number of Credits: 10% of the 15 credit module
Electronic Submission Deadline:
Tuesday 3rd May 2022 @ 11am
Autograder and Canvas

Learning Outcome(s): Gaining experience writing large programs from a specification with no initial code. Implementing data structures such as binary search trees and graphs. Using recursion.

1 Overview

"It's not what you know, it's who you know."
— LinkedIn, probably

It's true in every business and it's true in academia! In this assignment, you will implement a social network that captures collaborations between researchers.

As ever, concentrate on each part of the assignment individually and try to get it working. **Do not pass onto the next step without getting the previous step working.** There are plenty of marks to be gained from each step, and you are best served with excellent early steps than a poor attempt at every step.

The key to success in this assignment is concentrating on a small number of classes at each step and not worrying about the remaining code (only its ADT). The major difference between this assignment and A1 is that you are not given any code. Not having code is both an advantage and a disadvantage. We will test your solutions with a hidden data set that our automatic program selects. To ensure top marks, make your solution as general as possible. You are allowed to add additional methods to the classes mentioned, unless specifically instructed otherwise. Do not add any attributes to your classes beyond those mentioned.

2 Steps to Complete A2

These steps to complete assignment two are ordered in increasing difficulty to help you complete the assignment. The first four steps are core and of similar difficulty. You should complete them in the specified order. The fifth step is an advanced step and should not be attempted unless the core steps have been successfully completed. The final step provides an even greater challenge. **Do not attempt steps 5 and 6 unless you have the first four core steps working.** After each step, you should save and backup your code.

The six steps together are worth 90%. The remaining 10% will be awarded for coding style. Javadoc is required for all your methods and classes and you should follow the style guidelines distributed in Lab Task 1. You will submit to Autograder **and** Canvas, as per the submission instructions at the end of this document. Failure to follow the submission instructions will result in a deduction of 3 marks from style. Submitting a format other than `zip` (i.e. a `rar` file) will result in a deduction of 5 marks from style. Please see the submission instructions at the end for further details.

2.1 Step 1: Make a Profile

For this step, you are asked to make the following class:

- `Profile.java`

This class represents a profile of a researcher in our social network and will contain the following attributes:

- Strings for `familyNames` and `givenNames` (each could contain multiple words)
- Date of PhD Award as three separate integers (`yearPhD`, `monthPhD`, `dayPhD`)

- String `email` address
- An `ArrayList<String>` of research interests called `researchInts`
- A data structure of previous research partners represented as references to `Profile` called `collaborators`.

You must choose an appropriate data structure to use for the collaborators from the standard libraries; you can change it during a later step if you think of something better. Do not implement your own data structure. We don't care how many times researchers collaborated or when, just that they collaborated.

The profile will have a constructor with all the above attributes passed as parameters **except** collaborators. They must be in the order listed above. The profile will have get and set methods for all attributes (interests are returned and stored as an ArrayList). It should not have a get and set method for the collaborators. You should provide a `String getDateOfPhD` method that returns a string with day, month and year in YYYY-MM-DD format.

A profile also has the following methods:

- `void collaborate(Profile p)` - adds a researcher to the collaborators
- `boolean hasCollaboratedWith(Profile p)` - indicates whether a researcher has collaborated with a specified other
- `int numOfCollabs()` - returns the number of researchers for this profile
- A `toString()` method that converts the profile to a string

The `toString()` method should only be used for debugging. It should print a short description that helps you develop and debug your code. It should not be used once the code is finished (but do not delete it).

Create a `ProfileMain.java` file and class with a `main` method. Use this to test your class. During testing, we use our own ProfileMain class to check your code—not yours. So make sure all of the methods are implemented exactly as specified above.

This part of the assignment is worth a total of **20 marks**. Please test your class to make sure it works.

2.2 Step 2: Make a Tree Node

In this part of the assignment, create a Binary Search Tree (BST) node. The name of this class will be the following:

- `BSTNode.java`

The `BSTNode` class will contain the following information.

- `Profile researcher` - a reference to a profile
- `BSTNode l` - a reference to the left child
- `BSTNode r` - a reference to the right child

The binary search tree node will have a constructor that takes a single parameter, `elem`, which is a reference to a Profile. It has no other constructors.

The `BSTNode` will have the following methods:

- `Profile getResearcher()` - gets the researcher's profile associated with this node
- `void setL(BSTNode l)` - sets the left child of `this` to `l`
- `void setR(BSTNode r)` - sets the right child of `this` to `r`
- `BSTNode getL()` - gets the left child of `this`
- `BSTNode getR()` - gets the right child of `this`

Create a `BSTNodeMain.java` file and class with a `main` method. Use this to test your `BSTNode` only. Again, we will use our own test class, so ensure your methods are written exactly as specified.

This part of the assignment is worth a total of **10 marks**. Please complete it as specified above. Please test your code to make sure it works as specified.

2.3 Step 3: Make a Binary Search Tree

In this section, we make a preliminary binary search tree and a method to add nodes to it. **If your BST is not written correctly you will not be able to satisfactorily answer the later steps.**

We start by making a binary search tree class. This class will be called:

- `BST.java`

The class contains a single attribute `BSTNode root` which is the root of the binary tree. There are no other attributes in this class.

This class has a single constructor `BST()`. There are no other constructors in this class.

The class contains a method called `void insertReseacher(Profile p)`. This method begins by constructing a BSTNode, placing a reference to the profile in it as it does so. If the root is null, this node becomes the root. Otherwise, it calls a private method, which you create, that is recursive.

This recursive method starts at the root of `BST` and recursively calls either `getL()` or `getR()` of the `BSTNode` depending on whether the researcher's family name comes alphabetically before or after the family name of the current node's researcher (respectively). A base case occurs when the node belongs as the left child and `getL()` returns `null`. A second base case occurs when the node belongs as the right child and `getR()` returns `null`. The tree retains its binary search property after the method is complete.

Please make a `BSTMain.java` class with a main method. This main method should test all methods in this class.

Hard code nodes and a tree in BSTMain.java to make sure that it is working properly before proceeding to the next step. If you don't do this, it will be very difficult to finish the assignment.

There may be a temptation in later parts of this assignment to add a getRoot method which returns the root of the tree. **Do not add a getRoot method to this class for any part of the assignment.** You will lose marks. The outside world should not need to know that your BST is created with nodes organised in a linked structure. You will need to add methods to this class to complete other stages to this class. However, all tree manipulations need to be done in the BST class and nowhere else.

This part of the assignment is work **15 marks**.

2.4 Step 4: Make a File Reader

A good solution to steps 1-4 will receive a much higher mark than a less good solution to everything.

If you don't get your hard coded tree working, you can still get a few marks by doing some things in step 4. This is the only time you can break the "all steps in order" rule at the beginning of the assignment.

Create a new class called

- `FileReader.java`

It has no attributes. Now, we make a method `public static BST readResearcherProfiles (String filename)` that reads the profiles from disk and populates the tree.

The format for a single line in the text file is as follows:

```
<Given Names>,<Family Names>,
<YYYY-MM-DD>,<email>,<interest 1>/
<interest 2>/.../<interest n>
```

Note that commas, and not spaces, delimit the fields. The date of PhD award is specified by DD, MM, YYYY, where DD is day, MM is month, and YYYY is year. Notice that forward slashes delimit the interests and there can be arbitrarily many. The above information should be on a single line of the file (i.e. one profile is one line).

You can download an example file with three researchers from Canvas called `researchers.txt`. To fully test your `FileReader`, and your other classes, you may need to add more researchers to this file.

You have flexibility on how you write your collection of methods. The series of methods that you create will do the following:

- Reads the contents of the file as specified.
- For each line of the file, creates a `Profile`
- For each `Profile`, it uses `insertReseacher` to place it in the correct place of the `BST`

It is strongly recommended that you use one `Scanner` to read lines in the file and one `Scanner` to read the tokens on the line.

Please make a `FileReaderMain.java` class that tests if your file reader works. The main method should construct a binary search tree, printing the nodes created as you go. You should handle any exceptions within your class: our (hidden) test file and class will not handle exceptions. This part of the assignment is worth **20 marks**. **Part marks will be awarded if you just print each profile to the screen and don't construct the binary tree properly.**

2.5 Step 5: Alphabetical order

Do not attempt this step until all of steps 1-4 work correctly. It is okay to submit an incomplete assignment without steps 4 and 5.

In this section, create a method in the BST class that prints the tree out in alphabetical order. This method will be called `printAlphabetical()`. You can do this efficiently by using one of the traversal algorithms discussed in class. You will print the names in the format:

```
<Family Names>, <Given Names>
```

Note the space after the comma. This is a common format for names in bibliographies.

Please make a class `AlphaMain.java` that tests if this traversal works. It should simply print the traversal out the the screen.

This part of the assignment is worth a total of **10 marks**.

2.6 Step 6: Find a research influencer!

Do not attempt this step until all of steps 1-5 work correctly.

If you have reached this step, congrats. This step is an advanced step and will be quite challenging.

You notice that we have not used the collaborators list yet. This is due to the fact that the collaborators list encodes a graph.

In this step, create a class called:

- `Graph.java`

Graph contains one attribute, a BST called `tree`, and no other attributes.

The constructor of `Graph` takes a reference to a binary tree and a String which is the name of a file that containing an edge list. The file contains the list of all academic collaborations between researchers, represented by their family names. Each line in the edge file represents one collaboration. The lines of this file are defined like this:

```
<name 1>,<name 2>
<name 2>,<name 3>
<name 3>,<name 4>
<name 4>,<name 1>
```

Note no space after the comma. For example, you might find:

```
van Wijngaarden,Archambault
Archambault,Astarte
```

The Graph constructor uses the edge list to populate the *collaborators* list of each `Profile`. The Graph is undirected which means that if `<name 1>,<name 2>` is present in the edge list there is an edge in both directions and the collaboration should appear for both researchers. If the same pairing appears in the file multiple times it should not result in multiple entries in `collaborators`. If researchers with `<name 1>` or `<name 2>` do not appear in the `BST`, an error should be printed to the console and the line skipped but the rest of the graph should be created. **We consider last names here as unique identifiers of researchers.**

Some researchers collaborate with a lot of people. Let's call them influencers. It can be advantageous to work with an influencer and join their network.

The final task in the assignment is to write the following method in the `Graph` class:

- `Profile findInfluencer (String familyNames)`

This should find, for the researcher in the graph represented by `familyNames`, a candidate influencer. This influencer is present in the graph and meets the following criteria:

1. has collaborated with the most people; AND
2. has not already collaborated with the initial researcher

If no suitable influencer can be found, the method should return the profile of the researcher supplied. You should handle any errors with appropriate exceptions.

In order to complete this task you are allowed to add methods to other classes if you wish.

Create a `GraphMain.java` class with a `main` method. Write your own `edges.txt` file using names from your `researchers.txt`. Construct a `Graph` and a `BST`. Call `findInfluencer` for a few different `Profiles` and print them to the screen for testing purposes.

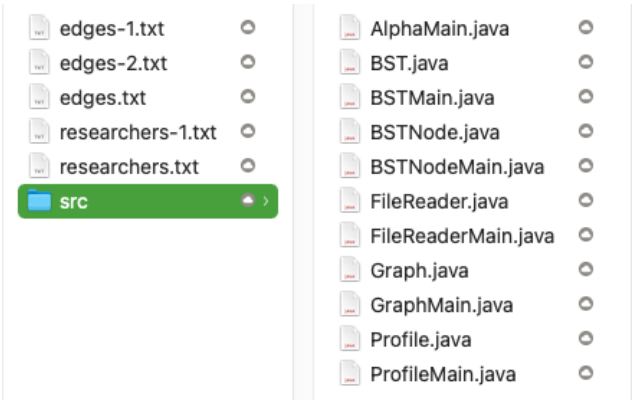
This part of the assignment is worth a total of **15 marks**.

3 Submission Instructions

Before you submit, remember to check your code for style. You must have Javadoc comments for every class and method and other comments as appropriate. You should follow the style guidelines given in Lab Task 1. Remember style is worth **10%** of your mark for this assignment.

Submission of the assignment is **electronic only**. In order to submit, create a `.zip` file of all your code, including your `Main` classes and all your input files as well, including the one given to you. All of your code should be in a directory named `src`. All of your input files should be placed one directory back from `src`. Input files should be named `researchers-n.txt` and `edges-n.txt` where `n` is a unique number, needed if you have multiple input files of each type.

Your directory structure looks like this:



(ignore the cloud symbols, they are from a cloud backup provider).

Submit this zip file to Canvas. The only format we will accept is `.zip`. Do not submit any other format. You will lose marks.

After submitting to Canvas **please check** that you uploaded the correct `.zip` file on canvas and that it is **readable and contains all files necessary for submission**. You are responsible for the correctness of your submission. Multiple submission attempts are allowed. If there is an error in your submission, please upload it again before the deadline. If all or some of the files necessary for your submission are not present or are unreadable, you will lose marks (or potentially get a zero). After the deadline, we will only use the most recent version of your assignment that was submitted to Canvas before the deadline. No other versions of your assignment will be accepted. No late submissions of the assignment are accepted without extenuating circumstances.

An autograder submission is required as part of this assignment as well. If you do not submit to autograder, the submission will not be considered valid and will receive a 0. Instructions for autograder are available through a link on the assignment page. You can access autograder here: `https://csautograder.swansea.ac.uk/`.

Feedback for this assignment will be given through autograder. When the assignment marks are released, you can receive your feedback there.

Best of luck! Happy coding!

Important Notice

By submitting to Swansea you acknowledge that you have read and understood the following:

By submitting this coursework, electronically and/or hardcopy, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at `https://myuni.swansea.ac.uk/academic-life/academic-misconduct`.

If you do not understand this text, please come see Troy during office hours.