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

Daniel Archambault

Number of Credits: 10% of the 15 credit module
Electronic Submission Deadline:
Friday May 10th, @11am

Learning Outcome(s): To gain experience writing large programs from scratch. Also, in the implementation of data structures such as binary search trees and graphs with some recursion.

1 Overview

In this assignment, if you complete it all the way to the end, you will implement a simple social network (without all the messy network communication stuff). This may sound a bit daunting, but once again, concentrate on each part of the assignment individually and try to get it working.

In this assignment, **do not pass onto the next step without getting the previous step working.** You can get a 1st in this assignment without doing step 6 at all. A 2i is possible without doing steps 5 and 6. The best strategy is to do an excellent job on earlier steps and not submit later steps. Please keep this in mind.

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 I don't give you any code or data. 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.

As with A1, I will lead you through this assignment indicating where you should start and providing a path through it. If you are having difficulty, please follow the steps described in Section 2 in order to complete the assignment.

2 Steps to Complete A2

These are the steps to complete assignment two. I have ordered them in increasing difficulty and in a way that ensures you can 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 three core steps working.** After each step, you should save and backup your code. This will prevent loss of work! Remember to regularly back up your code!

All six steps are worth 90%. The remaining 10% will be awarded for coding style.

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 person in our social network. A profile is kind of like a Facebook profile and will contain the following information:

- Last Name
- First Name
- Date of Birth (day, month, year) as three separate integers
- Town of Residence
- Country of Residence
- Nationality
- Email address
- An array of interests represented as strings

- An array or ArrayList of friends represented as references to `Profile`

The profile will have a constructor that can construct a profile with all the above attributes passed as parameters **except** the friends list. The profile will have get and set methods for town, country, nationality, and interests (interests are returned and stored as an array). It should not have a get and set method for the friends list. It will only have get methods for date of birth and name. You should provide a `getDateOfBirth` method that returns a string with day, month and year all as integers.

A profile also has the following methods:

- `void addFriend (Profile p)` - adds a friend to the friend list
- `Profile getFriend (int i)` - gets friend i from the list
- `int numOfFriends ()` - returns the number of friends 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 will create and copy in our own profile main class to check your code. 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 as specified.

2.2 Step 2: Make a Tree Node

In this part of the assignment, create a binary search tree node. The name of this node will be the following:

- `BSTNode.java`

The `BSTNode` will contain the following information.

- `Profile data` - 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, data, which is a reference to a Profile. It has no other constructors.

The `BSTNode` will have the following methods:

- `Profile getProfile ()` - gets the 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.

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. **Make sure its done well before proceeding to other parts of the assignment.**

We start by making a binary 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 insertProfile (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 if the name of the profile alphabetically comes before or after the name of the current BSTNode. 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.

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.

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

2.4 Step 4: Make a File Reader

A good solution to 1-4 will receive a much higher mark than a not so 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. Now, we make a method public static BST readProfileSet (String filename) that reads the profiles from disk and populates the tree. I would strongly suggest you break the code to read the profiles into several logical methods. If you only have one or two methods, you will lose marks.

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

```
<Last Name>,<First Name>,<DD>,<MM>,<YYYY>,<Town>,
<Country>,<Nationality>,<email>,<interest 1>;
<interest 2>;...;<interest n>
```

Note that commas, and not spaces, delimit the attributes. The date of birth is specified by DD, MM, YYYY, where DD is day, MM is month, and YYYY is year. Notice that semicolons 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 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 insertProfile 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. We will test your code with a test file that remains hidden from you.

This part of the assignment is work **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 that prints the tree out in alphabetical order. This method will be called printAlphabetical ().

As you have created a binary search tree, alphabetical order is simply an in order traversal of this tree. We have discussed in order traversals extensively in class. Implement this traversal, **printing out the name of the profile only**, as you visit each BSTNode.

Please make a AlphaMain.java class 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 me some friends!

Do not attempt this step until all of steps 1-5 work correctly. It is okay to submit an incomplete assignment.

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

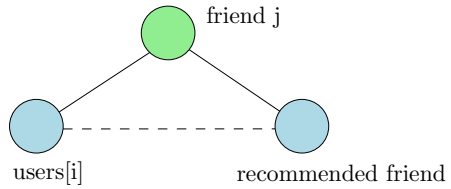

Figure 1: Friend recommendation procedure.

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

In this step, create a class called `Graph`. Graph contains a list of `Profile` instances which should all be in your binary search tree.

The constructor of Graph takes a string which corresponds to a filename and and a reference to a binary search tree. The file contains the list of all friendships between profiles. The lines of this file are defined as follows:

```
<name 1>,<name 2>
```

The names are two names in the BST. We assume that names are unique. If the name does not appear in the `BST` construction of the graph should end in error.

The only attribute in Graph is a BST. The constructor of `Graph` takes a reference to this binary tree and a file name to the file that contains the edge list. It uses the edge list to populate the friend list of each `Profile`.

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

- `BST[] friendRecommendations (Profile[] users)`

The parameter `users` is an array of profiles. An array of `BST` is returned containing all friend recommendations for each user in the users array. A friend is recommended if users[i] has a friend j who has a friend that is not friends with users[i] (figure 1).

This process is called *triadic closure*. It is an old concept in social networks and is often used as part of friend recommendation in social media.

Create a `GraphMain.java` file and class with a `main` method. Construct a `Graph` and a `BST`. Call `friendRecommendations` and use the reference to the created BSTs to print them out to the screen using `printAlphabetical ()`.

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

3 Submission Instructions

Submission of the assignment is **electronic only**. In order to submit, create a `.zip` file of all your code including your input files. Ensure all of the **data you submit** is in a **directory called data** which is in your source directory. Include all the Main files for each step and every class you wrote. The only format we will accept is `.zip`. Do not submit any other format. You will lose marks.

In order to submit to Blackboard, click on the red link in the assignments folder and follow the instructions to submit your zip file. Happy coding!