Social Networks in Great Britain

CS4624

Virginia Tech
Blacksburg, Virginia 24061

5/10/22

Group Member:
Javan Smith

Client:
Professor David Radcliffe

Instructor:
Dr. Edward A. Fox
Table of Contents:

Executive Summary / Abstract . . . . . . . . . . . . . . 3
Introduction . . . . . . . . . . 4
Requirements . . . . . . . . . . 5
Design . . . . . . . . . . 6
Implementation . . . . . . . . . . 7
Users’ Manual . . . . . . . . . . 8
Developer’s Manual . . . . . . . . . . 13
Lesson’s Learned . . . . . . . . . . 16
Acknowledgements . . . . . . . . . . 18
References . . . . . . . . . . 19

Table of Figures:

Figure 1 Individual Information Search . . . . . . . . . . 8
Figure 2 Individual Information Search Results . . . . . . 9
Figure 3 Get Associates Search . . . . . . . . . . . . . 10
Figure 4 Get Family Members Search . . . . . . . . . . 11
Figure 5 Get Family Members Search Results . . . . . . 12
Figure 6 Timeline . . . . . . . . . . . . . . . . . . . 16
WebsiteSocialNetworksBritain is a project designed to create a HTML website from an input of forty thousand XML records. These XML records describe historical British and American figures and their connections to each other through family, associates, and organizations. The client has built these records and wishes them to be viewable on a website. Per specifications, the website must be built using no third-party services. The final Java program consists of two parts: reading the XML records into a data structure and providing the ability to search the according to appropriate criteria. The program uses a binary search tree to store the forty thousand records and creates a pop-up menu to select the records the user wishes to view. The pop-up menu allows the user to search through the records and find previously unseen connections between the historical figures.
Introduction

This report describes the WebsiteSocialNetworksBritain project, henceforth SNiGB. The report is broken down into sections describing different aspects of the project. The client for the project is Professor David Radcliffe, an English professor at Virginia Tech. He has built about forty thousand XML records and wants them to be transformed into indexed and searchable records, presentable through HTML.
Requirements

The project’s requirements are unique due to the nature of the purpose for the website. Because the client wishes these records to be stored long term, no third-party service such as React, Node.js, Express, MySQL, etc. are allowed. The end goal is to build a program that can stand the test of time and will not need to be continuously updated. The forty thousand XML records need to be transformed into indexable and searchable HTML records.
**Design**

The project needs to be new user approachable. Searches need to be near instantaneous. The data structure needs to be searchable in under $O(n)$ time, with a goal time of $O(\log n)$. The user interface is in the form of a pop-up menu, with dropdowns that act as search specifiers (see Figure 1). Each XML file is stored in an object with parameters that correspond to the information in each file. There are two supporting objects to store specific types of information for a record.
Implementation

The program is built using Java. I decided to use a Binary Search Tree for the main data structure. While not the fastest compared to a HashMap, I have the most experience with trees, and a balanced Binary Search Tree will search in an average case of \( O(\log n) \). The menu is built using a JFrame with added JButtons, JLabels, and JComboBoxes. Each dropdown filters results from the Binary Search Tree to show in the next dropdown. The XML files are stored in Record objects that hold the necessary parameters to fully hold the information given. A Family object is used to represent each Record’s family, and a Value object is used to store name key pairs corresponding to the full name of a Record and its XML ID, respectively.
Users’ Manual

In order to launch the program, all that is required is for the “Social Networks in Great Britain.jar” file and the “SNiGB_Files_August” folder to be in the same directory.

Upon launch of the program, the user will see a menu with dropdown choices.

![Pick search parameters](image)

*Figure 1: Individual Information Search*

Each dropdown menu will update the menu below. Make sure for every search that each dropdown has been chosen. The first dropdown allows the user to pick either a specific letter or to search every letter. The letter corresponds to a record’s last name. The second dropdown allows the user to pick between three search options: Show individual information, Get Associates, and Get Family. The search options will change the output for the website. If a user
decides to search for a specific individual, the “Choose the record” dropdown will appear, which will have a list of potential records based on the letter chosen to search. Once the search bar is clicked, the user’s choices will be processed, and the website will be displayed with the results.

Figures 2 shows the results form what is shown in Figure 1.

Figure 2: Individual Information Search Results

Figure 3 shows the results with the search option for “Get Associates” chosen.
Instead of showing just the individual information, the website will display the associates for the chosen record, in the format of the previous output. These are marked from each other using dashed lines. If the record has no associates, a simple “No known associates” message is displayed on the website.
Figure 4 shows the search type for “Get Family Members”:

![Get Family Members Search](image)

*Figure 4: Get Family Members Search*
Choosing this option will show the family members for the chosen record if they are in the records database. If the chosen individual does not have any family members, the default family members are shown instead; see figure 5.

![No Known Family Members with XML Records](image)

Parents:
- Joseph Edward Acton
- Eleanore Grifin Berghe von Trips

Spouses:
- Zoe d'Albon

Siblings:
- Lady Mary Anne Acton
- Henry Acton
- Francis Acton
- Isabella Dachenhausen

Children:

*Figure 5: Get Family Members Default Search Result*

The “Close the Program” button will immediately terminate the program.
Developer’s Manual

File structure outline:

    Backend.java

Backend.java is the main class for the program. Its main method initializes a Binary Search Tree given in BST.java, extracts the current working directory, and sets up the path to the data files stored in “SNiGB_Files_August”. If the XML records need to be moved, line 44 in Backend.java is where you can update the location of the directory. The program then builds the BST by calling a method readDirectory with the path. This is due to the structure of the SNiGB directory: it holds folders labeled by alphabet (A, B, C, etc.), and inside those folders are the XML files. The readDirectory method calls readXMLFile for every XML file inside the directory that doesn’t start with a “.”. Doing so ignores the macOS version of the XML files. After line 151, the BST is filled, and the pop-up menu is built using a JFrame and adding buttons, labels, and dropdowns. Each dropdown has an action listener attached to update the other dropdowns in sequence to apply selections. On line 264, the search button is created. Because of the restrictions for no servers, the website is loaded manually in every sense: line 270 contains the bare outline, and the search button works by replacing “$body” with the content to replace based on the selections of the dropdowns. Once the logic is complete in the if/else statements, the program creates a “template.html” file that is used in the call for desktop.open() on line 330. That is the line that opens up the website in the browser. The rest of the methods are support methods for the searching logic and website building logic explained in the documentation.
**BST.java**

BST.java is the file that contains the data structure for the program, a Binary Search Tree. The BST is unbalanced by default, but can be balanced in linear time by calling balance(). The BST is not generic and uses a BinaryNode for its leaves. Its main search methods are on lines 56 and 66. Line 56 is the method to find an individual record, used for the “Show individual information” option. Line 66 is the method to find every record in the tree which last name starts with a given letter. This is used when an individual is not picked, and in populating the individual dropdown search after choosing a letter. The BST does not support deletions.

**BinaryNode.java**

BinaryNode.java describes the leaves used in BST.java. Each BinaryNode contains a Record.java element and pointers to its left and right leaves. The rest of the methods are getters and setters for the element and its two pointers, and a toString method.

**Record.java**

Record.java contains the information for an individual XML record. Its parameters are as follows: xmlID (String), fullName (String), birth (String), death (String), sex (String), bio (String), nationality (String), bibliography (ArrayList<String>), origin (String), family (Family), associates (ArrayList<Value>), organizations (ArrayList<String>), surname (String), forename (String), prefix (String), and suffix (String).

These parameters include all of the relevant information from an XML file. Every parameter has a getter but no setter, as once the record is read in Backend.java it shouldn’t change.

**Family.java**
Family.java contains the information for a Record's family. It contains four parameters, all ArrayLists of type Value, one for: parents, spouses, children, and siblings. The record is initialized empty and is only populated through specific adders for each parameter. Each parameter has a getter.

Value.java

Value.java consists of a name and key pair used for specific data structure types in Record.java and Family.java. The name is for a XML file’s full name, and the key is for the XML file’s ID (the file name). The parameters have getters and setters.
Lessons Learned

Figure 6: Timeline

Figure 6 shows the timeline for the project. The most important lesson I learned throughout the project is to make sure you understand what the client’s potential restrictions are. As seen on the timeline, the first two months were basically wasted as I did not realize I could not use frameworks such as React. This revelation forced a pivot in early April, leading to a one-month scramble to convert work and salvage what I could from the previous work.

Another problem I ran into was in late March / early April when I decided to pivot to Java. I had previously run a Java website using a Java Applet back in high school. These allow you to run client-side programs on browsers in an all-in-one solution. When I went to start outputting the results of the program to HTML, I discovered that the Applets had been deprecated back in 2016, leaving me to scramble for other solutions. Its replacement was a service called Java Servlets. Unfortunately, they required third-party server hosting to run, which due to the requirements of the project I was unable to use.

Future Work

The future work planned for this project includes implementing a network graph, a tough feat due to the restrictions applied. The project will also be hosted on Newman Library’s archives, which could lead to complications which will need to be addressed. Finally, if the restrictions are
changed a bit, the website could be changed to use any type of interactivity available in modern Java programs. For example, if Java Servlets are enabled, the website could be revamped to no longer be static after clicking the “Search” button, and to contain direct links to records. Doing so would also allow the presentation of the HTML to be more aesthetically pleasing through CSS files.
Acknowledgements

I’d like to acknowledge my client Professor David Radcliffe, and my instructor Professor Edward Fox.
References


https://doi.org/10.7294/14849748.v1

