Data Structures and Algorithms 1

CA Exercise 2 – Elections Information System

The objective of this CA exercise is to create an application for storing and retrieving information on Irish political elections. The application should include both hashing and sorting functionality, and provide a JavaFX graphical user interface.

The application should allow/support the following:

- Add a new politician to the system.
    - Name, date of birth, political party (if any; Independent if not), home county (e.g. Waterford) and an image/photo (as a URL) are among key data to store.
- Add a new election to the system. The system should support general, local, European and presidential elections.
    - Election type, location (e.g. county for locals), year/date, and number of winners (or seats; 1 or more) are among key data to store.
- Add a new candidate to an election, and record their total votes in that election. Only politicians on the system can be candidates in any election. Note that while politicians might have a current party membership (e.g. Fianna Fail) then may have stood for a different party (or as an Independent) in previous elections, and this affiliation should be appropriately recorded at the election level.
    - Note that you can exclude and ignore second preferences etc. as used in Ireland's actual STV voting system. Just use a simple vote tally approach for this project.
- Ability to edit/update/delete politician, election and candidate information.
- Ability to search or filter elections by type and/or year.
- Ability to search or filter politicians by (partial) name, party and/or location.
- Listings of search results (for elections and politicians) should be appropriately sorted depending on the chosen search parameters/options. Sorting could be by: politician name, party, year (ascending or descending), election type, etc.
    - You should provide for sorting of results in at least 2 different ways.
- Ability to view results of a given search, specifically politician and election details.
    - Politician details will provide their name, show their photo, party affiliation, etc. It will also show a list of elections they have been a candidate in.
    - Election details should show the election type, year, etc. It will also show a list of candidates sorted in descending order based on the votes they got in the election. The top/winning candidate(s) should be somehow highlighted depending on how many winners/seats were available.
- The system should support some level of interactive "drill down" for additional detail or navigation/browsing.
    - For instance, searching for "Fine Gael" politicians should provide a list of all politicians in Fine Gael; one politician could then be clicked on to see more information specifically on that politician; that detail could include a list of elections they previously stood in as a candidate; clicking on an election in that list opens up full details on the election, including a list of all candidates in that election alongside their votes; one of those could be clicked on to see their full politician details; and so on.

- The system should support some form of persistence whereby the internal data is saved to a file on exit, and reloaded for use on the next execution. You could use e.g. binary files, XML files, plain text files, or anything else that provides an image/snapshot of all internal data.
    - There is no need to look to databases or anything like that. A single snapshot/image file is fine for our purposes.

Notes

- This is a team CA exercise. Teams consist of 2 students only. Students should find/choose their own partners.
- You will have to demonstrate this CA exercise via Zoom (on the same call) and you may be interviewed individually on various aspects of it. You are expected to be able to answer all questions on all code individually (so make sure that you understand all code, including the parts that your teammate worked on).
- This CA exercise is worth approximately 45% of your overall module mark.
- As with CA Exercise 1, you cannot use any existing Java collections or data structures classes (e.g. ArrayList, LinkedList, or any other class that implements the Collection interface or any of its children – if in doubt, ask me!). You essentially have to implement the required data structures and algorithms from scratch and from first principles (in line with the module learning outcomes).
    - You are free to reuse any of your generic code from CA Exercise 1.
    - You can only use a regular array for your hash table(s).
- You also cannot use any sorting or searching methods provided by Java in e.g. the Arrays or Collections class (or any third party library equivalents). You should implement any searching and sorting routines from scratch. Again, this is a learning exercise, and in line with the module learning outcomes.
    - Note that you cannot use a bubble sort either (as this is the full example provided in the course notes, and I don't want you to just copy-paste that). Use any other sorting algorithm except a regular bubble sort for this reason.
    - Note also that you cannot just depend on JavaFX components to do the sorting for you. While some components might facilitate this you also have to manually sort results as part of this project too.
- You have to use hashing as part of the project to avoid (excessive) linear/sequential searching. For instance, a custom hash function should be used for retrieving details on a specific politician or election when searching. Again, you have to provide your own implementation of hashing.
- You have to provide a JavaFX based graphical user interface for the system. Exactly what your interface looks like, though, is up to you.
    - You can choose to implement a command line interface if you wish, but you will lose the marks allocated specifically for the JavaFX GUI.
- Remember that the key point of this CA exercise is to demonstrate knowledge and proficiency with hashing and sorting in particular. Keep this in mind!

Indicative Marking Scheme

- Appropriate custom ADTs for politicians, elections, etc. = 10%
- Create/add facilities (politicians, elections and candidates) = 10%
- Edit/update/delete facilities (politicians, elections and candidates) = 10%
- Search and listing facilities (multiple search options; politicians and elections) = 10%
- Sorting of search results/listings (politicians, elections and candidates) = 15%
- Hashing for individual search/lookup (politicians and elections) = 10%
- Persistence facility (saving and loading data) = 10%
- JavaFX graphical user interface = 10%
- JUnit testing (minimum of 6-8 useful unit tests) = 5%
- General (commenting, style, logical approach, completeness, etc.) = 10%