Mini Project 2

By: Andrew Rooney, Anthony Ma, Mustafa Khairullah

CMPUT 291

Title

1. General Overview and User Guide

Our project allows a user to retrieve and query information from an XML (".xml") file containing email records. This can be performed by the user via the following steps.

The first phase can be completed by writing the following command into the terminal, where "yourXMLFile" is the XML file containing the email records the user wishes to interact with. Note that this "yourXMLFile" must be in the project directory.

```
cat <yourXMLFile> | python3 parse.py
```

This command will create 4 text documents, emails.txt, dates.txt, terms.txt, and recs.txt. Each file contains key-value pairs interpreted from the XML file. The user can then perform the next phase by typing the following command into the terminal.

```
./buildIndeces.sh
```

This command will sort each text file created by parse.py, and then creates B+ tree indexes for emails, terms and dates, while also creating a hash index for records, all created indexes will be under the newly created directory "idx." The user has now completed the setup needed for the interface to run properly, the user can now access the interface by typing the following command into the

```
./start.sh
```

From here, the user can now retrieve information by entering queries that conform to the valid input format specified on eclass. Invalid input format can be entered but will not return any results.

2. Algorithm design

To evaluate queries that conform to the specifications on eClass, we employed a top down parsing algorithm known as recursive descent parsing. This algorithm allowed us to easily conform to the context-free grammar specified on eclass. For our recursive descent parse, we first used lexical analysis to break the user's input into tokens, the lexer would take care of extraneous whitespace and split each input down to . We would then invoke the parser object try to match a particular token to a variable at the top level of the grammar, and then we would go into a deeper level in the grammar by calling a function, and then within that function we try to match subsequent tokens to the grammar, we repeat this until we are at the deepest level of the specified grammar. At which point we will construct a variable to obtain data from the indexes using the dbbsb3 module.

If the user requests an exact match, we would iterate through the index and look for all IDs that had an exact match to the user. If the user had requested a range search, we would find the first entry that matches the user's specification, and then iterate and return all entries in the index before (less) or after (greater) the first entry (the first entry is excluded if the user wants an exclusive range search), since we the index is sorted by buildIndeces.sh.

Once all the ids are collected, we add them into a result set, if the user requests multiple queries, we intersect the ids returned by that query with the result set. We then use the ids as keys for the recs.idx index, and we run a nested loop were we loop through every id, and we find all emails from recs.idx that have each particular id as it's key.

The XML parser from part one also used a recursive descent parser, the parser object runs in O(n) time, where n is the number of lines in the XML input.

3. Testing

Each phase in the specification was tested with the primary intention of conforming to the desired logic flow, and appropriate responses to bad input, and other edge cases. The components were tested against these criteria.

- With normal input, are the correct data entries being returned to the user
- Does the interface reject or handle bad input that violate the context-free grammar that was supplied on eClass?
- Are the input queries that the user supplies case insensitive, does the program still run properly and output the desired results regardless of casing?
- During lexical analysis, are the correct tokens being produced for interpretation?
- Do multiple queries correctly return an intersection of the results of the multiple queries?
- Does having extraneous whitespace in the query entry affect the results?
- Does extended versions of a keyword return the same result as an abbreviated one?

The team tested each function by these conditions at the end of the project, When an item from the eClass list of questions was completed, it was unit tested by the creator following (generally) the specifications aforementioned. Then the tested code was pushed to GitHub, and the group members ran their own tests on edge cases they could think of, making improvements as needed.

4. Group Work

We used GitHub for this project, and each partner pushed their changes and work on project into the GitHub repository after a work session, we informed each other of the changes we made via git commit messages and Discord.

Andrew's contributions totalled about 12 hours and included design and implementation of the XML parser in part one and the file sorting and index file creation in part two. Andrew also designed the query parser for part three, given that he is the only one on the team with experience with parsers - and has implemented a recursive descent parser before.

Anthony's contributions to the project include implementing some productions from the query grammar in the query parser for part three, in addition to some augmentations on the user interface. Total time spent 5 hours.

Mustafa's contributions to the project include implementing some productions from the query grammar to the parser in part three. Total time spent 5 hours.

All three members tested their own functions and applied the criteria specified in the testing function above.

The breakdown of the work and contributions made to the project are a general breakdown and are not completely accurate, as all 3 members helped and coordinated with the other 2's work, supplementing their own understanding of the project. Each member shared various ideas and possible improvements to both the program and the design frequently, alongside useful resources found online.

Much of the project was completed when we met up in person, which allowed for the most convenient and efficient level of coordination. The rest of the project was completed on each member's own time and in group voice calls between the members.