Email Lookup App

General Overview and User Guide

Email Lookup App is a command-line information retrieval system. This application allows a user to search for email records matching various search criteria. Multiple criteria can be combined to further narrow down the search. Supported search criteria include the email addresses involved, the send date, and words in the subject or body. Users can choose to have full details about matching emails returned, or just a brief summary which includes the row number and subject. The query grammar is detailed in the following sections, all queries are case-insensitive.

To build the database, run:
`python3 phase1.py < file-to-process.xml`
`bash phase2.sh`

To start the application, run:
`python3 phase3.py`

Changing between brief and full output modes

The output mode can be changed by entering `output=brief` to switch to the brief summary mode or `output=full` to switch to the full details mode. This application will start in the brief summary mode.

Searching by dates

A user may search for emails that were sent on a certain date by prefixing the date they wish to search for with `date:`. There can optionally be whitespaces around the colon. The user can also search for a range of dates by replacing the colon with `>`, `<`, `>=`, `<=`. Following the prefix, a date in the `YYYY/MM/DD` format can be entered.

Example: `date > 2019/09/09`

Searching by email addresses

A user may search for emails by the email addresses involved. The prefix, `from:`, `to:`, `cc:`, or `bcc:`, searches for email addresses in the from, to, carbon copy, and blind carbon copy fields respectively. There can optionally be whitespaces around the colon. Following the prefix, an email address can be entered.

Example: `bcc: abc@x.yz`

Searching by terms

A user may search for emails by words found in the subject or body of an email. The prefix here works similarly to searching by email addresses, `subj:` or `body:` searches for the term in the subject and body fields respectively If the user wishes to search in both subject and body fields, the prefix can be omitted. Additionally, there is an optional suffix `%`, which acts as a wildcard. For example. `option%` would find terms like `options` and `optionally` in addition to the word `option`. Note that terms shorter than three characters cannot be searched.

Example: `subj: urgent`

Searching with multiple conditions

A user may decide to combine multiple search criteria described above. This can be done by having all criteria entered on one line, separated by at least one whitespace.

Example: `subj: urgent gas leakage date < 2010/10/10`

In this example above, emails matching the following criteria will be returned:

- Has the word `urgent` in the subject,
- Has the work `gas` and `leakage` in either the subject or body, and
- Been sent before `2010/10/10`

Software and Algorithms Design

Phase 2 is simply a Bash script, piping is used to chain commands to reduce disk writes. Phase 1 and phase 3 are written in Python and follow the DRY, or Don't Repeat Yourself principle. Code that is used multiple times is extracted to functions, similarly, large functions are broken down to smaller ones to ensure readability.

User inputs are parsed sequentially using regular expressions, we also experimented with a recursive descent parser, but unfortunately, that ended in a disaster. A global array stores row numbers that are currently matched and are used to iteratively narrow down the search. This is done by intersecting the existing global array with new result arrays. By not storing the full rows during the search, a large amount of memory can be saved. In the worst-case scenario, three copies of row numbers array need to be stored in the main memory: the existing row numbers array, the new row numbers array, and the temporary array used when computing the intersection of the two other arrays.

For evaluating partial or range queries, we first position the cursor to either end of the range then retrieve entries one by one until we reach the other end of the range. This avoids scanning the entire database looking for results. The running time to position the cursor is `O(log(n))` and the running time for retrieving results is `O(m)`,

where `n` is the total number of entries in the index and `m` is the number of matched results. We always use the most appropriate index for maximum performance.

When the global row numbers array becomes empty, meaning that there are no results, the rest of the query is skipped in order to further improve the performance.

When displaying the final result, the full row for each matching email is retrieved one by one and processed either by extracting and displaying the subject or displaying the full details. The full row is quickly discarded after being processed, so only one full row will be stored in the main memory at a time.

Testing Strategy

We decided to do manual testing for this project since that seems to be the easiest and least time-consuming. It worked fairly well for our last project so we expected it to work well this time as well.

Just like last time, the test cases that were created following the BDD, or Behavior Driven Development principle. We performed testing during development as well as a final testing on the lab machines before submission. This ensures that our code is compatible with the lab environment.

Each test case includes inputs and expected output, test results are recorded and tracked using GitHub and fixes are verified by another team member. Including the test cases given on eClass, about twenty test cases were created in total. The test cases achieve full branch coverage of our application.

Work Breakdown

We used a private Git repository to manage the versioning of our application, the repository is hosted on GitHub for ease of access. The GitHub issues feature was used to keep track of things to be done and Facebook Messenger was used for quick communications and discussions. We also regularly meet on campus to work on the project together, we spent about an equal amount of time on this project, about 8 hours each. A quick summary of each member's responsibilities are as follows:

- Arun Woosaree:
    - Project setup, speed optimization for part 2, phase 3 draft (skeleton code, initial parsing), report
- Hai Yang Xu:
    - Repository setup, phase 1, phase 3, report
- Tamara Bojovic:
    - Phase 2, testing