Daniel Han
Michelle Aubin
Ryan Kang

Mini Project 2 Design Document

General Overview

Our system, Information Retrieval System (IRS), uses the Berkeley DB library utilizing its ability to operate on files and indices to store and locate data using key terms. This was accomplished through three phases of development.

In phase 1, an ".xml" file is read through the terminal and processed through four different functions. Each function, except for the "create_records" function, searches for its respective strings within the file and outputs them into a ".txt" file. Each string that is outputted is followed by the row ID they were found in. The "create_records" function outputs all information in the ".xml" file into a ".txt" file. To execute phase 1, type "python3 phase1.py name_of_file.xml" into the terminal.

Phase 2 sorts the data from the .txt files, putting the sorted data back into their respective .txt files. Then each of the .txt files are ran through the given "break.pl" program, and the output from this is piped to the db_load command to create the indices. To execute phase 2, type "sh phase2.sh" into the terminal.

Phase 3 contains an interface in which a user is prompted to enter a query. In this prompt, the user is given four options: Type in "output=brief" to get the row ID and title of the query result, "output=full" for the full record, "exit" to exit the program, or a query that adheres to the "Query Language Grammar". To execute phase 3, type "python3 queries.py" into the terminal.

Query Algorithm

First we split the input by spaces and ":" into a list. We then parse through the list as a queue, popping out elements as we create queries. This creates a 2D list Queries, where each element is a query. Time complexity is O(n), where n is the number of words in the query.

All log(N) are in base b, where b is the average number of keys per page and N is the total number of keys in the database.

We go through queries in a loop. We look at the condition for the query, then go into a separate function to search for its corresponding rows in the appropriate index. We return a set of row numbers from the helper function, which is then intersected with the set of row numbers from preceding queries. Once we have gone through all the queries, we pass the set

of row numbers to the searchRecords function to retrieve the key-value pairs of all of the matching records for our queries. Then, we format the data from these key-value pairs and print the results.

To search for terms we set the query to the term and find all rows that have that term. When subj or body is specified we return the respective rows, but when it is a general term search we union the result of body and subj search. This runs in O(log(N) + L), L being the number of rows with the term. For wild card searches we use set_range on the term because it does not require a complete key, and then iterate through next until the prefix is not found anymore. This runs in O(log(N) + L), L being the number of rows with the prefix.

To search for a specific date we pass the date as a cursor set and return all rows that satisfy it. This runs in O(log(N) + L), L being the number of rows with the date. For range search less than date, we set cursor as the first tuple then add tuples through next until the row is greater than the date. This runs in O(L), L being the number of rows less than the date. For range search greater than date, we set the date given. If the date is not in the database, we increase by 1 day until a date is found. We then add all tuples until the end of the database. This runs in O(log(N) + L + X), L being the number of days between the date requested and the first date in the database, and X being the number of rows greater than the date.

To search for emails we open the “em.idx” index and set the cursor to the first key-value pair that has the email from the condition as the key. In a loop, we add the value (the row number) to a set and use the next_dup() method to get the next key-value pair with the same email as the key. We return the set of row numbers to the main function. This runs in O(log(N)).

To search through the records, we open the “re.idx” index and loop through the set of row numbers for our query. In the loop, we set the cursor to the key-value pair that has the row number as the key and append the value (the full record) to a list. We return the list of records to the main function. This runs in O(R) where R is the number of row numbers for our query.

<u>Testing Strategy: Performed by all members</u>

Testing for phase 1 and phase 2 was mainly confirming the correctness of the contents of each file as well as the correct type of file, “.txt” and “.idx” respectively. This was done through the usage of the small data set “10_data.xml”. Necessary testing that applied to phase 3: Functionality and correct display of information for “exit”, “output=brief”, and “output=full”. Correct display for inappropriate query formatting and no-result queries. Correctness of user-defined functions and testing of range cases for certain functions (dates, terms). For large data sets, pick a random email and try to

locate it through different combinations of subj, body, to, from, date, etc. For date searches test dates outside the range of the database, and dates not in the database. Look for the difference between >= / > and <= / <.

Group Work Break-down

The breakdown of the work was mainly divided depending on which information each individual was handling. Daniel mainly handled functions regarding dates, Michelle handled functions regarding emails and Ryan regarding terms. Michelle took on the additional role of creating phase 2 as well as handling some aspects of input and output for the queries in phase 3. Ryan created the query preprocessor in phase 3 to divide the input into individual queries. All group members contributed to debugging and testing for the entire program. Version control was established through GitHub. Communication was done in-person as well as through the usage of group chats.

Group Meetings: 7 Hours

Michelle:

- Individual Time: 6 Hours
- Progress made: emails.txt (Phase 1), Phase 2, searchEmails, searchRecords, helped with handling input, formatting output for full (Phase 3)

Daniel:

- Individual Time: 5.5 Hours
- Progress made: dates.txt & recs.txt (Phase 1), searchDates(Phase 3)

Ryan:

- Individual Time: 7 Hours
- Progress made: terms.txt (Phase 1), searchTerms, query preprocessor, helped Daniel with searchDates, formatting output for brief (Phase 3)