CMPUT 291 Mini Project 2 Report

by Helen Aquino, Onn Qazilbash, Cenab Bora Batu

A. Overview

Before the main program is run, the four .txt files would be sorted and then used to create indices, which the main program will use to search for data. The program first prompts the user to input one or more queries, where the input will then be processed into separate queries. Each query will be processed through different functions, depending on what it asks for (term search, range search, etc). The resulting list will contain the resulting review IDs which will be used to then retrieve the proper data, based on the requested output (full or brief). The program then displays the output, then prompts the user again for input, continuing the entire process until the user wishes to stop.

Quick Guide

The four .txt files (scores.txt, pterms.txt, rterms.txt, reviews.txt), Makefile, phase1.py, and phase2.py must exist in the same directory.

To build the databases, enter the following commands in the command line:

"make format_files"

"make build_index"

To run the phase2.py, enter the following command: *"python3 phase2.py"*

B. Algorithm Description

Phase 1

- Makefile: sorts the four .txt files; calls on phase1.py to parse the text files; creates four .idx files from the sorted .txt files using the db_load command (hash index for reviews.txt, B+-tree indices for the remaining three files)
- phase1.py: parses the text files and creates

Phase 2 (phase2.py)

main()

- init_databases(): initializes the four databases, sets flags, and opens the .idx files
- init_cursors(): initializes the fours cursors
- In a while loop, creates an empty list 'results' to pass onto other functions that will store the review ids of the search results
    - User is prompted to input queries
    - Queries are then processed by process_text() and stored in queries variable
        - process_text(): contains a list of valid operators (`"="`, `":"`, `"<"`, `">"`); for each operator, in a while loop, it checks wherever the operator is present and parses it to a valid query (depending on which operator is present); queries are then stored in variable all_queries and returned

- queries are then passed to compute_results(queries, results): for each query in queries, program checks which operator is present and passes the query to one of the following functions:
  - '<' - range_search_bigger(query, results):
    - Price and Date: Price/Date is not in any index so the scores index was used to retrieve record ids. Using the range search starting at score = 0, each record was retrieved and its resulting price/date was compared. If product price/date is greater than what the user-specified, we store the record_id
    - Score: Since score has its own index, range search was useful to obtain scores greater than what the user-specified. The record_ids were then stored.
  - '>' - range_search_smaller(query, results):
    - Same algorithm as above, but with opposite comparisons
  - ':' - term_search(query, results): handles 2 cases: whether the query begins with 'pterm' or 'rterm';
    - Case 1: 'pterm' - the term is passed onto pterm_search(term, results)
    - Case 2: 'rterm' - the term is passed onto rterm_search(term, results)
    - In both cases, either function handles both partial matching and exact term matching,
      - For partial matching the % is removed form the string and the program goes through the respective .idx file (pt.idx for pterm_search and rt.idx for rterm_search), decodes each line and checks if there are any phrases/words that begin with the term/phrase (for multi-word searches) and appends the resulting review id to results list
      - For exact term, the term is encoded and then the program goes through the respective .idx file to check for any matches and appends any resulting review id to results list
  - '=' - stores the requested output mode in variable OUTPUT
  - If the term is a single word with '%' following it: single_term_search(query, results): handles 2 cases - partial matching and exact term matching;
    - if handling partial matching, the '%' is removed from the string; the program first goes through each of the decoded lines in pt.idx do check if any terms begin with the query string, and if not, the program then checks through decodes lines in rt.idx;
    - for handling exact terms, the query is encoded and then goes through both pt.idx and rt.idx for any matches;

    Each match's review id will be stored in a list rev_ids which will be appended to the results list

        - If query doesn't match any of the previous cases - invalid_input(): informs the user that the query contains an invalid input and closes the program
    - A variable ids stores the results of function intersect_tables(id_sets): The intersection of each tuple in the list will be taken in order to get the shared review ids among all queries
    - ids is then passed to function print_tables(ids): Given a set of review ids, this function will format and print the output. Depending on the specified output, the function will provide a specific list of fields and indices to be used to print out for each corresponding review id
    - continue_query(): asks the user if they want to continue search; if no, while loop ends
- close_connection(): closes all databases and cursors

C. Testing Strategy

The general idea for our program testing is to use the provided query examples to the given text files to ensure the correct data was being retrieved from the system. In addition to testing with the provided databases and indexes, the program was also tested with the 10k and 20k records database to ensure low querying times.
Issues found: Functions for price/date range queries were time-consuming and non-efficient as database size increased.

D. Group Work Strategy

The team members communicated throughout the project using Slack. That was where we stated what needs to get done at the moment. Our method of splitting up the tasks was very flexible; each member worked on what was the most immediate task to tend to. We kept our files in a Github repository so that there would be no conflicts in code.

Task Breakdown

Onn (~15)	Batu(~9h)	Helen (~9h)
Contributed to Phase 1 - Created phase1.py to ready .txt files for database creation - Created makefile Contributed to Phase 2 - Wrote base code (small functions, connect/disconnect to db) - Implemented term search (w/ partial matching) - Contributed to range search functionalities	Contributed to Phase 2 - Handling bigger range search + smaller range search - Fixing mild inefficiency issues - Contributed to debugging and testing	Contributed to phase2.py - Implemented handling single term queries + partial matching - fixing mild inefficiency issues Wrote up the report