CMPUT 291 Mini-Project 1 Report

Jarrett Yu, Michael Huang, Nicholas Serrano

Overview

Introduction

The goal of this assignment is to create a simple DBMS program which, given premade indices, accepts simple queries and returns a set of data that matches the search terms. The program specifically is made to emulate a peer selling site. A user can search for certain keywords and specify dates, prices, locations, and category.

Setup and Operation Instructions

The code has compliance with Java 1.7 and newer. Run “javac Main.java” to compile the program, then run “java Main <file>” to create the txt files.

The bash script load.sh should be run to sort and generate the .idx files.

The command “java -classpath .:/usr/share/java/db-5.3.28.jar Main” should be used to run the part3 on the lab machines. All .idx files should be present in the same directory and the program should be compiled using the “make” command.

Design

Design Overview

Part 1

Part 1 reads a .txt file from standard input and outputs the specified files in the spec. The program reads the input file line by line as strings. Each line is parsed with regex and written to the correct output file.

Part 2

Part 2 consists of a Bash script which runs the linux sort command on each of the output files from part 1. Then the script runs the given perl script on the sorted files which are loaded into a .idx file using db_load simultaneously.

Part 3

Part 3 is split into 3 main parts; the query parser, the database searcher, and the database manager. The query parser handles receiving, parsing, organizing, and printing of queries. Once given an input line it will figure out what kind of query should be used, and extracts the information needed to perform the query. These are then processed one expression at a time by calling a method dedicated to that search type. The results are assembled into a set which are later intersected to collect the results that satisfy all search terms. These are then printed from the same program.

Query Breakdown

Part 3’s query parser works by breaking down a query to its independent binary expressions. It does not perform ranged sorts since although it does improve runtime speed it is more difficult to implement. When an inequality is the operator the parser requests a query that continues until it reaches the end or beginning of the indices and returns all values retrieved.

The query’s can be categorized into two forms, equality searches and ranged searches. Equality searches run in linear time since they iterate through the ad’s document and searches for matches. It then immediately adds the required data since all data is stored in the value. Searches take a longer time and is around a $NlogN$ runtime. This is because it initially searches for a key and stores its aid in a temporary variable. These are then looked up in the ad.idx table.

The runtimes of term searching, and ranged searches could be significantly optimized but weren’t due to time constraints. If a table of aid’s were shared then the code could have looked up the information required with a reduced dataset, cutting out many of the results before searching. Additionally, proper ranged searches were not implemented due to an increased complexity in the parsing stage.

Testing

Initial testing was performed on the 10-record case file. Each query type was testing using these data sets to ensure correctness. Then, the 100k record case was considered to see how the program’s runtime would be affected.

The methodology for debugging the 10-record case file was to run total searches in addition to conditional searches. This told us if our code properly went through all options in the file and if it would return the expected results. Print statements were added in various blocks to check where the code would access under specific conditions or if a line wasn’t printed.

It is noted that query optimization would greatly improve the runtime. When running the 100k file we tested using single search terms to get an approximate measure of the runtime. Given our skipping of implementing query optimizations and refinement we found that more queries would increase the runtime in a linear manner.

Production Organization

The project was logically split into multiple parts with each person handling a specific task and assisting other members when they were waiting on other member’s progress. Most of the work for the assignment was done as a group in person. Code was shared and stored on a github repository. The closer interaction of the modules required more direct communication with each group member.

Michael handled phase 1 and 2 as well as the initialization of the databases for part 3. He was focused on getting data into the form for part 3. Phase 1 took longer than expected because of regex errors. Took too long to figure out how to use db_load and find the correct options to build each idx file. Approximate time spent: 22h

Nick handled phase 3 and did most of the work on creating the queries which are used in the query parsing. There were 5 different types of queries to handle. Nick would access the databases and come up with algorithms to efficiently find the requested query results. First implementation was very slow for huge databases and would sometimes take a few minutes to fetch results. Later implementations use Berkley dB functions and take advantage of the way databases are sorted. Approximate time spent: 20h

Jarrett also handled phase 3 and did most of the system framework, query parsing, debugging, and most of the report. He also built the search by term part of the query handler. Later, he rewrote most of the Query Handler to improve the runtime and code quality. Approximate time spent: 22h